20 projets à réaliser dès 10 ans

Programmer avec Arduino

EN S'AMUSANT

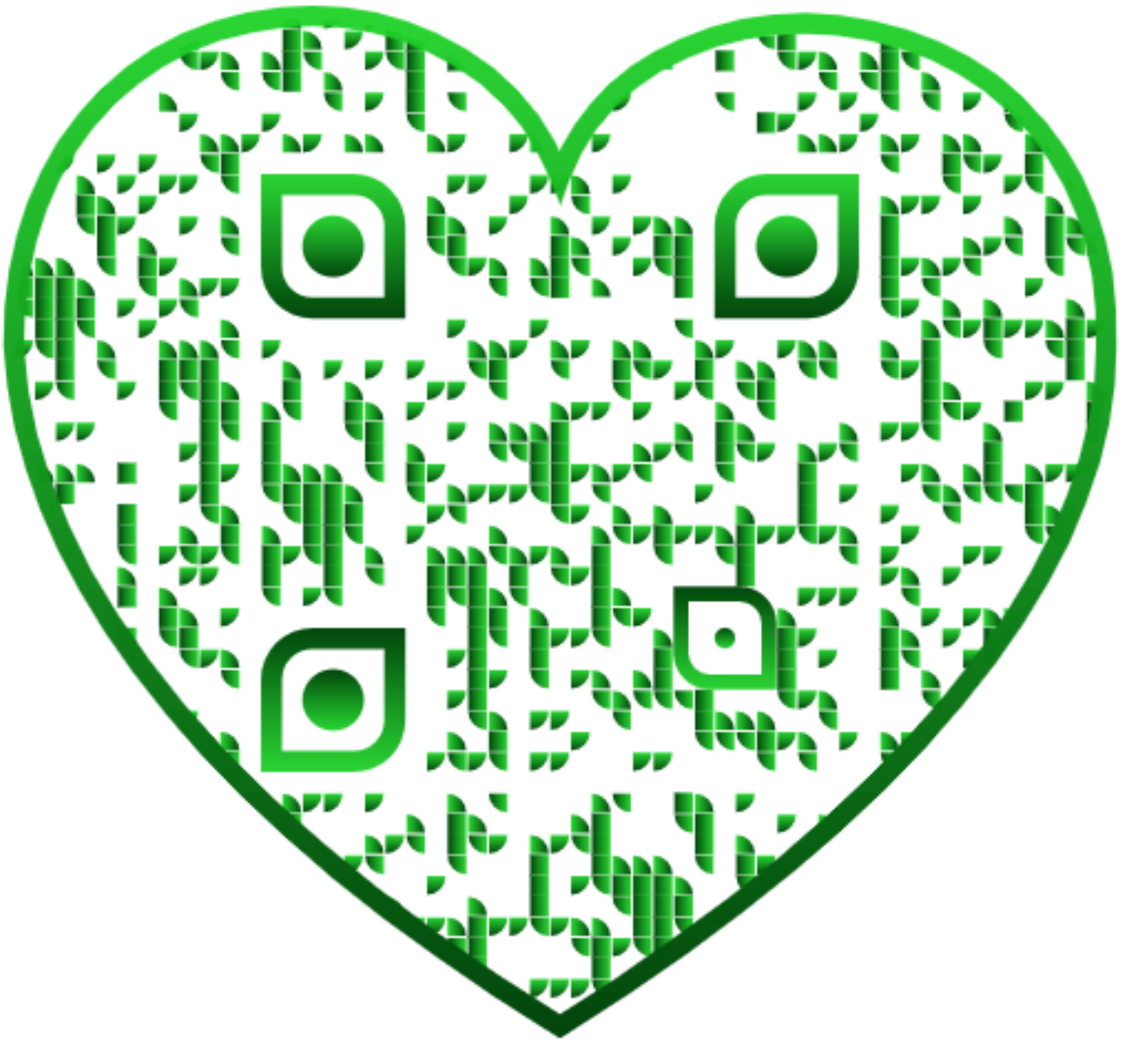
Installer et tester le micro-contrôleur

Maîtriser les vraies bases de la programmation

Développer des projets ludiques en étant autonome

OLIVIER ENGLER

pour les nuls



Programmer avec Arduino en s'amusant

pour les nuls

Olivier Engler

FIRST Interactive

Programmer avec Arduino en s'amusant pour les Nuls

Pour les Nuls est une marque déposée de Wiley Publishing, Inc

For Dummies est une marque déposée de Wiley Publishing, Inc

Collection dirigée par Jean-Pierre Cano
Maquette : Pierre Brandeis

Edition française publiée en accord avec Wiley Publishing, Inc.
© Éditions First, un département d'Édi8, 2017
12 avenue d'Italie 75013 Paris
Tél. : 01 44 16 09 00
Fax : 01 44 16 09 01
e-mail : firstinfo@efirst.com
Internet : www.editionsfirst.fr
ISBN : 978-2-412-02387-7
ISBn numérique : 9782412028001
Dépôt légal : mai 2017
Imprimé en France

Tous droits réservés. Toute reproduction, même partielle, du contenu, de la couverture ou des icônes, par quelque procédé que ce soit (électronique, photocopie, bande magnétique ou autre) est interdite sans autorisation par écrit de Wiley Publishing, Inc.

Cette œuvre est protégée par le droit d'auteur et strictement réservée à l'usage privé du client. Toute reproduction ou diffusion au profit de tiers, à titre gratuit ou onéreux, de tout ou partie de cette œuvre est strictement interdite et constitue une contrefaçon prévue par les articles L 335-2 et suivants du Code de la propriété intellectuelle. L'éditeur se réserve le droit de poursuivre toute atteinte à ses droits de propriété intellectuelle devant les juridictions civiles ou pénales.

Ce livre numérique a été converti initialement au format EPUB par Isako www.isako.com à partir de l'édition papier du même ouvrage.

Introduction

Bienvenue dans le monde de l'invisible, de l'intouchable, de la matière molle, du soft-ware !

Nous allons vraiment nous amuser à prendre puis à garder le contrôle de quelque chose de trop étrange : ta propre capacité à insuffler du raisonnement dans un bébé-cerveau fait avec du sable.

Quand tu écris d'habitude, c'est pour te relire (fonction de mémoire) ou bien pour que quelqu'un d'autre te relise (fonction de communication).

Dans ce livre, tu vas apprendre à écrire des histoires qui vont être vécues par un circuit électronique comme sa vraie et unique réalité.

Sans programme, un ordinateur est l'objet le plus inutile que l'on puisse fabriquer.

Avec un programme, c'est l'outil le plus utile, le plus polyvalent et le plus complice de l'homme qui puisse s'imaginer.

L’état d’esprit du livre

Presque tous les livres consacrés au phénomène Arduino se veulent pratiques. Le lecteur a très envie de savoir construire toutes sortes de montages, et son impatience est supposée. On est guidé dans le montage, dans le soudage, on va piocher par téléchargement un exemple déjà écrit et testé, on le transfère sur la carte, et hop ! ça marche. Gratification immédiate.

Et après ? À quel moment auras-tu appris à expérimenter, à devenir autonome, à avoir de nouvelles idées si tu n’as pas fait très lentement les premiers pas ?

Je te propose ici de t’émerveiller devant de premiers montages très simples au niveau matériel, mais de plus en plus évolués au niveau programmation. Ce livre privilégie le code. La matière obéit, le code dirige !

Après cette mise en route, tu pourras envisager de poursuivre avec un environnement de programmation comme ceux des professionnels : langage C, C++, Java, JavaScript, Python, atelier Eclipse… Mais d’abord les bases.

Sécurité !

Un défi que je me suis donné concerne le fer à souder. Dans deux projets, savoir manier cet outil ou avoir dans les parages quelqu’un qui sait est plus confortable, mais on peut faire sans. Aucun branchement à une prise électrique 220 V non plus. Châtaigne, go home !

Résumé de l’histoire (plan du livre)

Ce livre propose quatre parties qui peuvent se faire en quatre semaines (ou quatre mois, mais quatre jours, ce serait stressant).

Semaine 1 : Mise en route

Dans le **Chapitre 1**, « Quel est cet animal ? » , nous découvrons l’objet à dompter en apprenant un langage. Cet objet est un microcontrôleur installé sur une carte électronique. C’est le seul chapitre que tu peux parcourir une première fois, t’arrêter lorsque cela devient corsé, puis y revenir (sans faute, hein ?) après avoir déroulé plusieurs autres chapitres.

Le **Chapitre 2**, « L’atelier du cyberartisan » , te guide dans la mise en place sur ton ordinateur d’un logiciel pour créer et modifier tes programmes puis les envoyer vers la carte.

Enfin, le **Chapitre 3**, « Le tour de chauffe » , explique comment récupérer et implanter les exemples du livre, découvrir le Moniteur série et charger un premier programme.

Semaine 2 : Parler et écouter le monde

C’est dans le **Chapitre 4**, « À l’écoute du monde en noir et blanc » , que nous faisons nos premiers pas dans l’art d’écrire un texte source en surveillant l’état d’une entrée numérique.

Nous enchaînons dans le **Chapitre 5**, « À l’écoute du monde réel » , avec les entrées analogiques, les types de données, ta première `fonction()` et les conditions. Nous ne sommes plus en vacances, mais c’est un feu d’artifice de nouveautés !

Pour se reposer les neurones, nous utilisons nos petits doigts pour construire des effets lumineux inédits avec des diodes LED dans le **Chapitre 6**, « Parler, c'est agir » . Ce ne sera pas par hasard que nous ferons générer des valeurs aléatoires.

Notre mikon va se montrer sociable comme jamais dans le **Chapitre 7**, « Donner une impulsion » , en produisant des valeurs analogiques. Pour clore la semaine, nous allons créer un jeu (de société) de devinette numérique.

Semaine 3 : Sons et lumières

Place aux modules. De l'intelligence concentrée. Le **Chapitre 8**, « Ta baguette magique » , explore la lumière invisible et ose aborder un tout petit peu la programmation orientée objets. Dans la trousse de base du programmeur, nous versons la structure **switch case** et les tableaux de valeurs. Rien que cela !

Avec le **Chapitre 9**, « Que personne ne bouge ! » , ce sera « sus aux intrus » et nous ferons du bruit… et de l'algorithmique.

Enfin, le **Chapitre 10**, « Le voleur de couleurs » , repeint le monde avec seulement du rouge, du vert et du bleu. Mais éteins la lumière, c'est mieux.

Semaine 4 : Le temps des lettres

Dans le **Chapitre 11**, « Le temps liquide » , nous affichons notre caractère et nous prenons notre temps pour le faire.

Nous repartons du côté obscur des ondes dans le **Chapitre 12**, « Un peu d'échologie » . Le mikon entendra ce que nous n'entendons pas. Et pour nous rassurer, nous terminons avec un instrument sonore très malléable.

L'**Annexe** réunit un mémento du langage Arduino et quelques pistes pour t'inviter à poursuivre l'aventure.

Pour réaliser les projets

Bien sûr, il te faut d'abord ta carte Arduino Uno ou une compatible. La vraie, celle de couleur « bleu Arduino » , vaut environ 20 euros. Les clones sont normalement d'une autre couleur (blanc, rouge, bleu ciel, vert) et moins chers. Si tu choisis un clone, fais-toi garantir par le vendeur qu'il est vraiment compatible.

Si tu trouves une carte nommée **Genuino**, c'est aussi la carte officielle, mais il y a eu une époque où les créateurs italiens de l'Arduino n'avaient plus le droit d'utiliser le nom en dehors des USA. Les choses sont revenues dans l'ordre.

Le tableau suivant présente les besoins en composants majeurs de tous les projets du livre. J'y ajoute un lot de pièces indispensables dans la réserve de toute personne qui a envie de réaliser des montages en électronique numérique.

Voici d'abord ces composants que tu vas pouvoir réutiliser d'un projet à l'autre :

- au moins une **plaque d'essai** ou de prototypage, une de taille normale (800 trous environ ; si tu le peux, prends aussi une de demi-longueur (400 trous) et une miniature (170 trous) (voir la fin du Chapitre 2) ;
- au moins 5 **résistances** de 100 ohms, 5 autres de 220 ohms, 5 de 470 ohms et au moins une de 1 kohm ;
- une dizaine de **fils** de connexion multicolores avec broches terminale mâle-mâle (des courts et des plus longs) ;
- une dizaine de fils de connexion multicolores avec broches terminale mâle-femelle ;

- quelques fils de connexion multicolores avec broches terminale femelle-femelle ;
- (en option) un **porte-pile** 9 V.

Le budget pour ces incontournables est d'environ 10 euros.

Je ne vais pas citer ces composants chaque fois dans les montages. Je vais supposer que tu en as toujours assez d'avance dans ton petit stock. En revanche, pour chaque projet, je cite le composant majeur, celui en général le plus onéreux.

Chapitre	*Projet*	*Qté*	*Composant majeur*	*Budget*
04	`Digit`	1	Bout de strap	0 €
05	`Analo`	1	Bout de strap	0 €
05	`Tempera`	1	Capteur de température **LM35** ou LM335 (+ rés. 1 kohm)	3 €
06	`Luciole`	1	Diode **LED** rouge, verte ou bleue (+ rés. 220 ohms)	0,50 €
07	`Binar`	-	idem Luciole	0 €
07	`Binar2`	1	=Binar1 + seconde diode LED (+ rés. de 220 ohms)	0,50 €
07	`Binar4`	2	=Binar2 + deux autres LED (+ 2 rés. de 220 ohms)	1 €

07	`Devinum`	-	idem Binar4	0 €
08	`BagMagik`	1	Capteur infrarouge de type **TSOP38238**	3 €
09	`Gardien`	1	Capteur PIR Velleman **VMA314** ou compatible	6 €
09	`Sonner`	1	Haut-parleur 8 ohms ou 32 ohms	3 €
09	`TestOreilles`	-	idem Sonner	0 €
10	`Tricolore`	1	Diode **LED RGB** anode commune (de préférence) ou cathode commune (+ 3 rés. 270 ohms)	2 €
10	`Arkenciel`	-	idem Tricolore	0 €
10	`VolKool`	1	Capteur de couleur Adafruit **TCS 34725**	10 €
11	`KristaLiq`	1	Module afficheur LCD de DFRobot **DFR0009**	11 €
11	`KristAlpha`	-	idem Kristaliq	0 €
11	`KristaProverbe`	-	idem Kristaliq	0 €
11	`Tokante`	1		10 €

			Module horloge RTC **TinyRTC** à base de DS1307	
12	`Telemaster`	1	Module à ultrasons **HC-SR05** (Velleman VMA306 ou compatible)	5 €
12	`TelemasterLCD`	-	idem Telemaster + afficheur de KristaLiq	0 €
12	`Taprochpa`	-	idem Telemaster + haut-parleur de Sonner	0 €
			TOTAL	55 €

Si ce budget est trop élevé, s'il faut enlever un des composants les plus chers, tu choisiras selon tes préférences. Tu pourras te passer de l'horloge DS1307 ou du capteur de couleur TCS 34725. Tu peux réaliser tous les projets jusqu'au Chapitre 9 inclus pour moins de 20 euros.

Une boîte de rangement translucide multicasier est très pratique pour y ranger toutes les petites pièces.

Je donne des conseils concernant les sites de revendeurs à la fin du livre.

La communauté Arduino

Quand tu entres dans le monde Arduino, tu viens rejoindre une communauté de millions de pratiquants qui se retrouvent sur les sites Web bien sûr, mais pas seulement : de nombreux locaux associatifs t'ouvrent leurs portes en métropole et en francophonie. Tu peux y obtenir de l'aide et, surtout, partager tes découvertes et trouver des partenaires pour réaliser quelque chose à plusieurs.

Une bonne adresse à fréquenter en premier est le forum en français du site officiel :

http://forum.arduino.cc/index.php?board=33.0

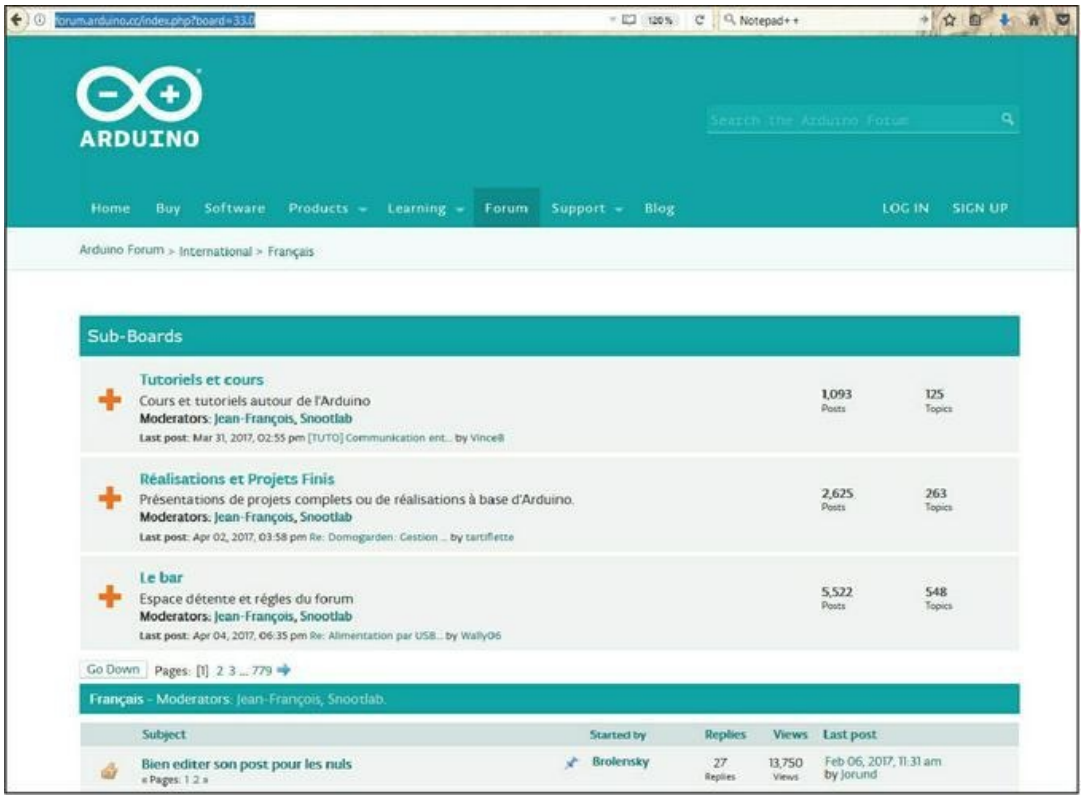


Figure 1 : Page d'accueil du forum Arduino francophone.

La valeur 33 est le préfixe téléphonique de la France.

Et maintenant, en avant toute !

Semaine 1 : Mise en route

Au menu de cette semaine :

Chapitre 1 : Quel est cet animal ?

Chapitre 2 : Les outils qu'il te faut

Chapitre 3 : Le tour de chauffe

Chapitre 1
Quel est cet animal ?

AU MENU DU CHAPITRE :

- **Il était une carte**
- **Il était un mikon**
- **Des langages pour raisonner**

En général, on dit qu'un être humain est constitué d'un corps et d'une âme, donc d'une pensée. Si tu n'avais pas de corps, tu ne pourrais sans doute pas te rendre compte que tu penses, donc tu ne pourrais sans doute pas penser.

Il en va de même avec un ordinateur : il se compose d'une partie matérielle (sa quincaillerie ou hardware) et d'une partie immatérielle (son programme ou software). Cela n'a rien d'étonnant : l'ordinateur a été inventé par les humains.

Ce livre se consacre à la carte Arduino Uno, et j'ai déjà indiqué dans l'introduction qu'il s'agissait d'une carte d'expérimentation construite autour d'un microcontrôleur (MCU, Micro Controller Unit). Est-ce différent d'un ordinateur ?

Dans un ordinateur habituel, il faut réunir sur une grande carte nommée carte mère de nombreux composants. Voici les plus importants :

- un processeur ;

- des barrettes de mémoire ;
- des circuits d'entrée-sortie (pour contrôler le disque dur, la carte graphique, la liaison Internet, les ports USB) ;
- une alimentation.

Un microcontrôleur réunit la plupart de ces composants sur une seule puce. Il est généralement bien moins puissant qu'un processeur d'ordinateur de bureau, mais il regroupe dans la même puce les trois éléments fondamentaux d'un ordinateur :

- un processeur pour traiter les données et faire les calculs (la CPU) – ici, c'est notre mikon ;
- un espace de stockage mémoire pour les instructions à exécuter et les données qu'il manipule ;
- des moyens pour communiquer avec l'extérieur, qui sont ses ports d'entrée et ses ports de sortie.

Au lieu de commencer par une visite guidée de la carte Arduino, je te propose de plonger d'abord dans la puce du mikon, le roi de la carte. Nous verrons sa « cour » , les composants qui l'entourent, dans un second temps.

Dans toute la suite du livre, pour éviter d'écrire sans cesse « microcontrôleur » , je vais baptiser notre nouvel ami « mikon » .

Ton mikon est un ATmega328

Un composant se distingue de tous les autres sur ta carte Arduino : c'est le microcontrôleur, pardon, le mikon. Il contient au moins un million de fois plus de composants que tous ceux que tu vois autour de lui sur la carte !

Ce composant est utilisé dans des millions de produits, des appareils ménagers aux gadgets électroniques, des alarmes aux machines-outils de l'industrie. Il se présente sous la forme d'un rectangle de plastique noir avec des pattes métalliques.

Figure 1.1 : Apparence externe du mikon ATmega328.

Protégée dans son boîtier noir se trouve une fine lamelle de silicium gravé d'environ 3 mm de côté : c'est elle, la puce. Toutes les pattes des deux côtés du boîtier (il y en a 14 de chaque côté) sont reliées à cette petite plaque de silicium.

La plaque contient des centaines de milliers de circuits électroniques élémentaires. La Figure 1.2 montre à quoi ressemble cette petite puce énormément agrandie. Je te rappelle que la taille réelle est de 3 mm sur 3 !

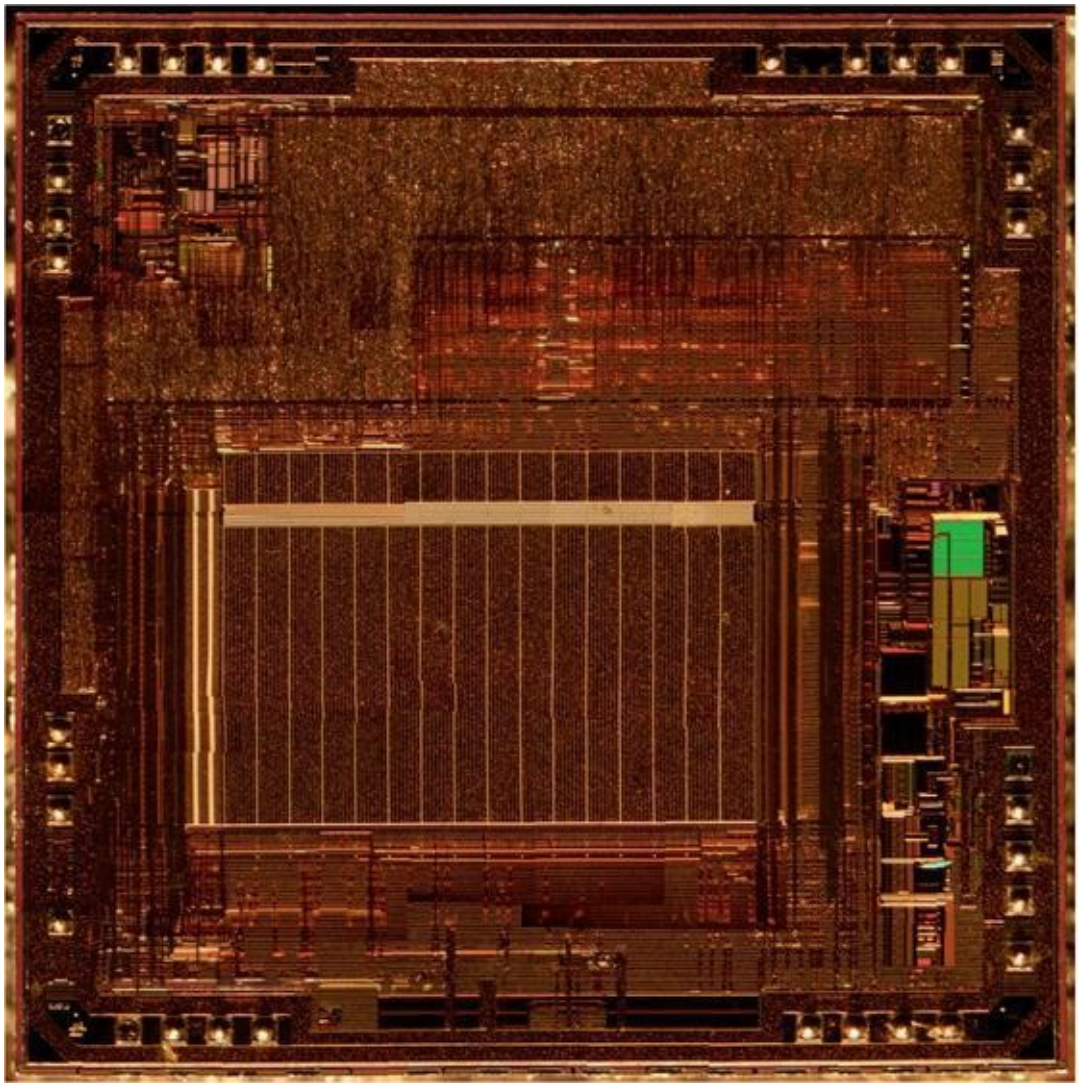


Figure 1.2 : Vue microscopique du circuit ATmega328.

Et voici sur la Figure 1.3 les ensembles de composants qui sont réunis dans le mikon. Je rappelle que dans les ordinateurs de bureau, la plupart de ces boîtes sont des composants séparés.

Il n'existe aucun objet dans l'univers connu qui possède une telle densité d'intelligence, pour un prix de 3 euros environ pièce. Tu peux par exemple acheter des poussins à 2 euros pièce pour te lancer dans l'élevage de poules pondeuses, mais les poussins ne sont pas des objets. Ce sont des organismes vivants très complexes. Bien sûr, ils exécutent aussi un programme, mais c'est leur programme génétique codé dans leur ADN.

Le mikon n'est pas un cerveau vide qui attend que tu y transfères ton premier programme. Dès sa sortie de l'usine, il sait faire un certain nombre d'opérations élémentaires (son jeu d'instructions), et il ne pourra jamais en faire d'autres.

La mission d'un programmeur consiste à décomposer suffisamment des opérations complexes pour qu'elles puissent être réalisées avec ces instructions élémentaires.

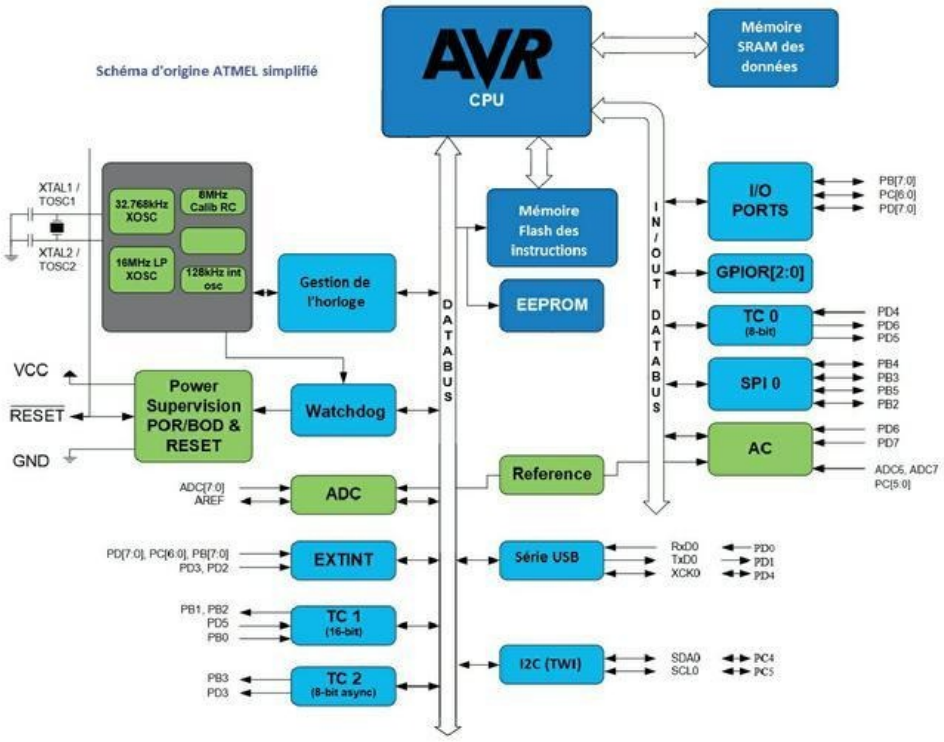


Figure 1.3 : Synoptique du microcontrôleur ATmega328.

Voici les principales catégories d'opérations que sait faire ton mikon (ainsi que tous les microprocesseurs actuels) :

1. **Opérations arithmétiques.** Addition, soustraction, multiplication, *etc*.

   Quand tu fais une addition de tête, tu dois souvent mémoriser un résultat intermédiaire quelque part (« et je retiens 3 »). Tu utilises pour cela ta mémoire à court

terme. La puce fait de même : elle a besoin de petits espaces de stockage dans lesquelles elle peut écrire le résultat intermédiaire de ses opérations.

Ces petits espaces s'appellent des *registres*. La puce ATmega 328 possède 32 registres. Chaque registre contient huit petits interrupteurs électriques (pour huit bits).

En fait, ces registres sont utilisés par paires sous forme de 16 registres de 16 bits. Nous verrons plus loin à quoi ils servent et ce que sont les bits.

2. **Opérations de déplacement.** Avant de commencer une opération, le mikon doit aller chercher les valeurs dans sa mémoire. Une fois l'opération terminée, il doit y ranger le résultat. Pour tout cela, il faut un espace mémoire en dehors des registres. C'est pourquoi le mikon possède toute une série d'instructions pour charger une valeur et stocker une valeur dans la mémoire de données (SRAM).

3. **Opérations de branchement.** La troisième catégorie d'instructions est la plus importante : elle réunit toutes les opérations qui font d'un processeur autre chose qu'un simple automate. En effet, grâce aux instructions de branchement combinées à des instructions de test, le comportement exact du programme peut varier d'une fois à la suivante. Je m'explique.

   Lorsque tu veux réaliser une recette de cuisine, tu suis dans l'ordre les étapes numérotées. Il n'y aurait aucun

intérêt à sauter de l'étape 1 à l'étape 4 puis à remonter vers les étapes 2 puis 3. Le fonctionnement est linéaire : tu vas du début à la fin.

Dans certaines recettes, tu laisses reposer une partie de ce que tu prépares pour confectionner, par exemple, une sauce qui sera ajoutée au plat. C'est une sous-recette. Quand la sauce est prête, tu reprends la suite de la recette principale.

Un programme dans un processeur est lui aussi constitué d'instructions qui se suivent. Le processeur les exécute l'une après l'autre au rythme d'une horloge (qui donne le tempo pour passer à l'instruction suivante). Cependant, tu peux décider avec une instruction de branchement de sauter certaines instructions pour continuer l'exécution un peu plus loin puis revenir, ce qui donne une souplesse infinie.

4. **Opérations de test.** C'est cette dernière catégorie d'instructions que tu combines avec la précédente pour décider de changer le comportement du programme en fonction d'une valeur qui peut être comparée à 0, ou à une autre valeur pour savoir si elle est égale, inférieure ou supérieure, *etc*.

Il existe une dernière catégorie d'instructions : celles qui servent à manipuler les signaux électriques bit par bit. Nous avons déjà assez de choses à découvrir pour les laisser de côté pour l'instant.

Au total, le microcontrôleur ATmega328 possède un peu plus d'une centaine d'instructions machine.

Un processeur est donc une machine qui utilise du temps, au rythme des tops de son horloge, et de l'espace (sa mémoire), pour modifier et créer des données. Les instructions de son programme sont stockées dans sa mémoire Flash et les données, dans une autre mémoire de travail (marquée SRAM sur le schéma suivant).

Voici le niveau de détail ultime pour les principaux constituants de ton mikon. La figure suivante montre uniquement les détails du rectangle AVR CPU et des deux boîtes de mémoire (Flash et SRAM) de la figure précédente. Petit, mais costaud !

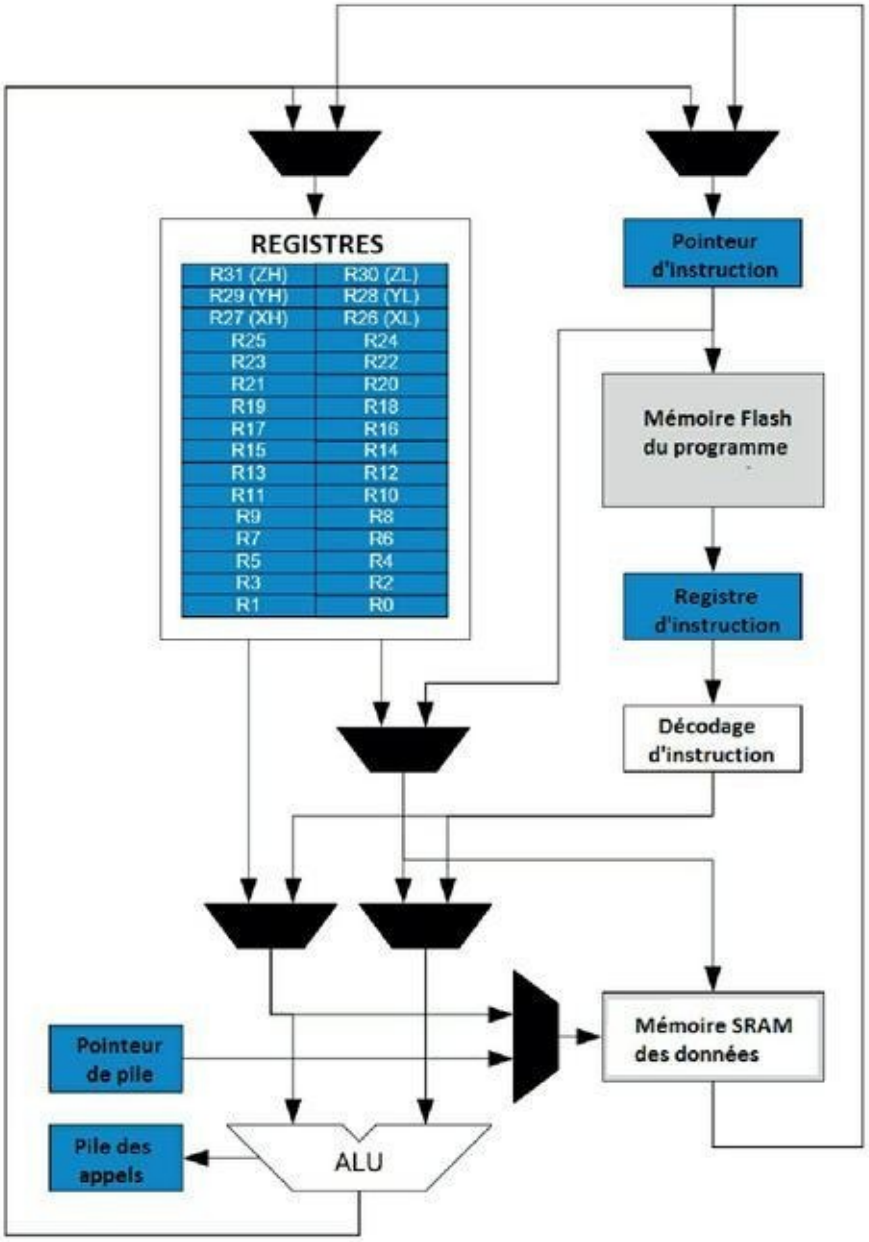


Figure 1.4 : Composants de l'unité centrale du mikon.

La mémoire du programme (mémoire Flash)

Lorsque le programme démarre, il va chercher la valeur se trouvant tout au début de cette zone mémoire. Cette valeur numérique représente la première instruction qu'il doit exécuter.

Une fois celle-ci exécutée (même pendant qu'elle s'exécute), il va chercher l'instruction suivante et ainsi de suite jusqu'à la fin du programme, avec éventuellement des branchements et des sauts, des chargements et déchargements de données et des opérations arithmétiques. Chaque action de recherche de l'instruction suivante se fait au rythme des tops d'horloge. Pour savoir où il en est dans l'avancement parmi les instructions, le mikon possède un registre mémoire spécial : le pointeur d'instruction.

C'est quoi son travail ?

Le cœur d'un mikon est un ensemble de circuits capables de réagir à des valeurs numériques (binaires) pour en créer d'autres. Ce travail est fait par l'unité arithmétique et logique ALU (Arithmetic and Logic Unit). Elle sait réaliser environ une centaine d'opérations différentes. Ce sont les instructions machine dont j'ai parlé plus haut.

Les résultats de ces traitements peuvent être rendus disponibles sur des broches de sortie de la puce ATmega qui sont disponibles sur les connexions marquées DIGITAL de ta carte Arduino. Sur chacune des broches de sortie, il peut y avoir soit une tension de 5 volts, soit pas de tension (0 volt). La combinaison exacte de 0 et de 1 sur les broches de sortie dépend entièrement du programme en cours d'exécution dans le mikon.

D’une certaine façon, le mikon fabrique de la dentelle de tension à partir de la tension continue 5 V qui lui sert d’alimentation électrique. C’est une dentelle très fine car les tensions sur les broches de sortie peuvent changer jusqu’à plusieurs millions de fois par seconde sur un Arduino Uno.

Le top de l’horloge

Le processeur fonctionne en progressant d’une étape à chaque coup d’une horloge, c’est-à-dire à chaque changement de tension entre 0 V et 5 V du signal d’horloge qui est stabilisé par un cristal de quartz. Tu peux voir le composant cristal sur la carte : il se trouve du côté gauche au milieu, dans un boîtier rectangulaire à coins arrondis. Si tu le regardes de près, tu devrais pouvoir lire la valeur 16 suivie d’autres chiffres. Cela signifie que ce cristal résonne à 16 MHz, c’est-à-dire qu’il oscille et change d’état 16 millions de fois par seconde.

Fréquence et gourmandise

La fréquence d’horloge de ton mikon est bien inférieure à celle des processeurs équipant les ordinateurs de bureau actuels. Ces CPU Intel ou AMD ont une horloge tournant entre 1 et 3 GHz (trois milliards de hertz), soit environ 100 fois plus vite que celle de ton mikon (16 MHz ou moins).

Mais les processeurs sont comme les cerveaux humains : plus tu réfléchis intensément, plus ton cerveau consomme de sucre, et plus souvent tu dois te recharger en énergie. Ton Arduino Uno consomme beaucoup moins que le processeur d’un ordinateur classique et chauffe donc beaucoup moins.

Lorsqu’il exécute des instructions, le processeur Atmel ATmega328 consomme environ 10 mA, soit 0,05 watts alors qu’un processeur

de milieu de gamme actuel consomme environ 1 000 fois plus (50 watts).

Mieux encore, lorsque le processeur ATmega bascule en veille, c'est-à-dire au repos (mais prêt à se réveiller s'il arrive quelque chose sur une broche d'entrée), la consommation descend à 0,1 µA (micro-ampère, encore 100 000 fois moins).

C'est ce qui permet d'utiliser des cartes Arduino pour toutes sortes de montages sur piles en domotique, en électronique de loisirs, dans les voitures, l'électroménager, les jouets, etc.

Ta carte, c'est une Uno

Reculons maintenant pour voir si notre mikon est bien entouré sur la carte. D'abord, je rappelle qu'il existe toute une famille de cartes Arduino, de la minuscule Nano à la grande sœur Mega 2560. Notre préférée est la Uno. La marque officielle est Arduino ou bien Genuino.

L'électricité statique tue les mikon !

Une mise en garde : en fonction des habits que tu portes, il est possible que tu accumules sans le savoir de l'électricité statique. Tu as peut-être déjà reçu une décharge en enlevant un pull ou un T-shirt. Pour éviter de détruire ta carte avant même d'avoir joué avec, touche par exemple un radiateur ou une poignée de porte, puis sors-la de son emballage.

Dans tous les cas, ne touche pas inutilement les contacts sur le dessous du circuit.

Tu peux prendre la carte en main en la tenant par les connecteurs en plastique en haut et en bas et en l'orientant dans le bon sens pour

pouvoir lire les textes. Que vois-tu ?

Figure 1.5 : Vue générale de la carte Arduino Uno.

- En haut à gauche, il y a un gros carré en métal. C'est, tu l'avais déjà deviné, la *prise USB*.
- En bas à gauche il y a un autre connecteur, arrondi celui-ci. Il servira plus tard à insérer un *jack* pour alimenter ta carte sans avoir besoin de garder le câble USB. En général, tu brancheras dans ce jack un adaptateur de pile 9 V.
- Entre ces deux prises, tu dois pouvoir détecter un tout petit circuit noir à gauche du rectangle ovalisé du cristal de quartz. C'est le *régulateur de tension*, qui permet d'alimenter proprement le microcontrôleur avec une tension toujours bien égale à

5 V et sans parasites.

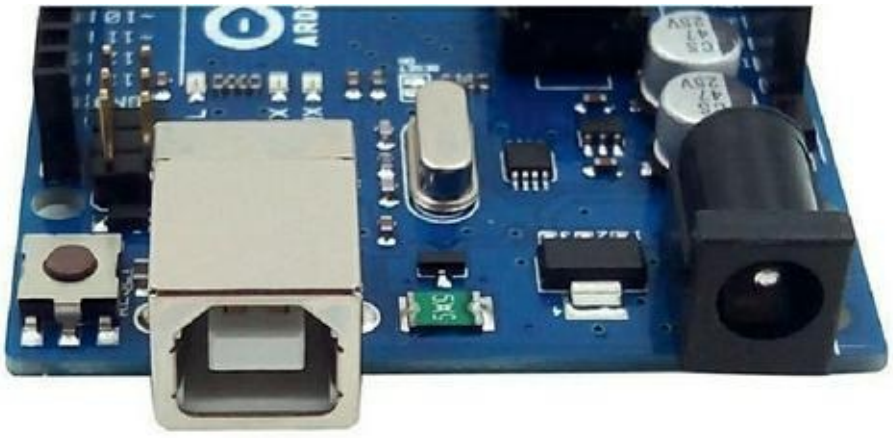


Figure 1.6 : Gros plan sur le connecteur USB et le jack d'alimentation.

» Tu remarques également deux cylindres gris clair en bas, à droite de la prise jack d'alimentation. Ce sont des *condensateurs* qui filtrent la tension d'entrée.

Avec toutes ces précautions, ton microcontrôleur sera alimenté avec de l'énergie de haute qualité, et il t'en remerciera.

En longeant le bord inférieur vers la droite, nous trouvons le connecteur noir des broches d'alimentation. Seules trois nous seront utiles, et dans tous les projets, d'ailleurs.

En les parcourant de gauche à droite (seules celles que nous allons utiliser sont en gras) :

» La première broche n'a pas de légende et n'est pas utilisée.

» La broche IOREF sert à distribuer à des cartes d'extension la même tension que celle utilisée par la carte Arduino. Elle fournit soit du 5 V, soit du 3,3 V.

- La broche RESET sert à reporter sur une carte d'extension le bouton RESET du coin supérieur gauche qui fait redémarrer le programme.
- La broche 3,3 V sert à alimenter des composants fonctionnant avec cette tension réduite au lieu de 5 V (maximum 50 mA).
- **La broche 5 V** est la seule sur laquelle tu peux obtenir le +5 V pour alimenter les composants. Elle est parfois marquée Vcc.
- **Les deux broches GND** correspondent au pôle négatif d'alimentation, c'est-à-dire à la masse, en anglais *ground*, abrégé en GND et parfois en Vss.
- Enfin, la broche VIN donne la même tension que celle qui est fournie en entrée lorsque tu alimentes la carte avec une pile ou un adaptateur secteur.

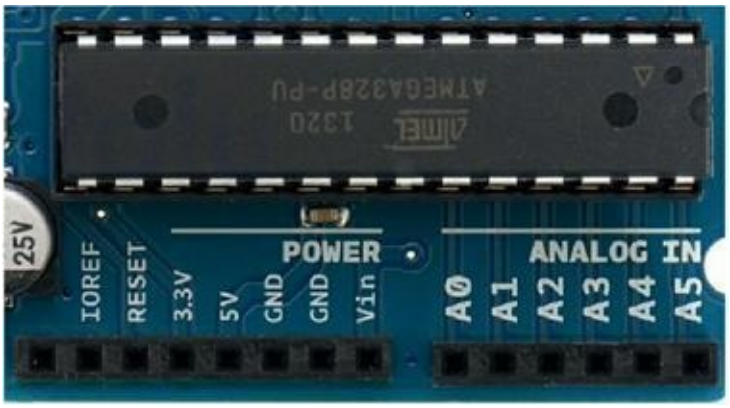


Figure 1.7 : Sorties d'alimentation et entrées analogiques.

Ne le brusque pas !

Ta carte Arduino et ton mikon sont des objets délicats. Il faut faire attention à ne pas trop demander au mikon. Ce qui pourrait facilement le détruire, c'est de lui demander de fournir trop de courant. Voyons ce qu'est un courant électrique.

Le courant, c'est de l'énergie qui se déplace. Quand tu vois l'eau couler dans une rivière, tu devines qu'elle va d'un point haut vers un point plus bas parce qu'elle est attirée par le centre de la terre et va finir dans la mer. Si tu installes un moulin sur la rivière, l'eau qui se déplace fera tourner le moulin. L'eau produira un travail parce que son déplacement représente de l'énergie. Il en va de même avec le courant électrique qui coule entre les deux bornes d'une pile ou entre les deux fils d'une prise électrique.

Si tu as des radiateurs électriques chez toi, tu sais qu'en réglant un radiateur au maximum, tu vas générer plus de chaleur. C'est tout simplement parce que tu fais passer plus de courant dans le radiateur. Ce courant est freiné par des résistances et c'est ce freinage qui crée la chaleur. Si tu règles le radiateur à moitié, il consommera deux fois moins de courant. Le courant se mesure en ampères (A). Pour un radiateur de 2 000 W en 220 V, la consommation est d'environ 10 A (2 000 divisés par 220).

Ton mikon est beaucoup, beaucoup, moins gourmand : chacune des broches de sortie ne peut fournir au maximum que 0,04 ampère, soit 40 milliampères (mA). L'ensemble des broches ne doit pas dépasser les 200 milliampères. Il faudra prendre les mêmes précautions en utilisant les broches en entrée : ne jamais faire passer plus de 40 mA par une broche de ta carte Arduino.

Suite de la visite

Le dernier connecteur en bas rassemble les six broches d'entrées analogiques à droite, repérées par la mention ANALOG IN (voir la Figure 1.7).

En remontant vers le bord supérieur de la carte, nous saluons le mikon au passage.

Le mikon et son embase

En général, car cela varie d'un fabricant à l'autre, le mikon se présente sous la forme d'un rectangle noir avec 14 pattes de chaque côté. Si tu regardes bien, tu dois voir que ce circuit est inséré dans une embase en plastique. C'est cette embase qui est soudée sur la carte. C'est une très sage précaution, car cela permet de changer uniquement le microcontrôleur au lieu de jeter la carte. Si par malheur tu le détruis un jour, il suffira de tirer doucement aux deux extrémités pour extraire la puce, puis en insérer une neuve. Cela épargne de devoir dessouder toutes ces petites broches ou de jeter la carte.

Nous arrivons enfin aux broches de sortie sur le bord supérieur, à côté de la mention DIGITAL. C'est sur ces broches qu'il y aura ou pas une tension de 5 V en fonction des opérations réalisées par le mikon (Figure 1.8).

Figure 1.8 : Gros plan sur les entrées/sorties numériques.

Le but du mikon n'est pas de travailler dans son coin, mais d'écouter ce qui se passe autour de lui puis de réagir en parlant à son tour. Voyons donc les entrées-sorties plus en détails.

Écouter et parler : les entrées et sorties

Tu sais maintenant que cette minuscule puce de 3 mm de côté est capable de réaliser, par exemple, 1 million d'instructions par seconde. Mais à quoi sert tout ce travail si tu ne peux pas en profiter ?

Il faut que les résultats puissent être renvoyés vers l'extérieur, et c'est à cela que servent les broches de sortie numérique.

Débranche si nécessaire le câble USB pour pouvoir tourner la carte Arduino dans tous les sens. Prends-la et observe bien le bord supérieur de la carte, là où se trouvent les deux connecteurs noirs, au-dessus de la mention DIGITAL PWM.

Regarde les chiffres marqués juste sous les connecteurs : ils vont de 0 à 13, ce qui fait 14 broches de sortie. La mention DIGITAL te rappelle que sur ces broches, il ne pourra y avoir que soit 0 V, soit 5 V. Ce sont des broches tout-ou-rien. Dans tes programmes, tu pourras écrire des instructions pour forcer par exemple la broche numérotée 7 (donc la huitième) à l'état haut, c'est-à-dire de manière qu'il y ait une tension de 5 V sur ce contact.

En français, on parle de « sortie numérique » , mais le terme anglais digital est tellement répandu que tu le verras souvent utilisé. Il faut savoir qu'en anglais, un chiffre se dit digit. En français, ce mot n'est utilisé que pour les empreintes digitales. Tu peux utiliser le mot « digital » , mais n'oublie pas que tu rencontreras le mot « numérique » si tu poursuis dans la voie de l'électronique.

Retourne la carte Arduino pour voir le côté pile. Oriente la plaque par rapport à ta source de lumière pour pouvoir deviner ce qui est gravé sur la surface. Tu dois voir des traits qui vont des broches du connecteur du haut jusqu'aux broches du mikon en bas. Cela montre que les connecteurs de la carte Arduino donnent directement accès aux pattes du circuit intégré, de façon bien plus confortable parce que tu peux directement insérer un fil dans chacun des connecteurs noirs.

Revenons aux légendes des sorties numériques (voir Figure 1.8). Quelques-unes ne se résument pas au numéro de la broche.

Rx et Tx sont en bateau

Les broches 0 et 1 sont suivies respectivement de la mention RX et TX. Ces deux broches servent à faire communiquer la carte avec ton ordinateur, comme elle le fait par la prise USB. Dans tes projets, dès que tu auras besoin d'afficher sur l'ordinateur des valeurs envoyées par le mikon, il ne faudra jamais utiliser ces deux broches 0 et 1 dans le montage, car cela générerait un conflit et un résultat bizarre.

Ces deux broches sont directement reliées (tu dois me croire) aux deux petites LED marquées TX et RX du côté gauche, un peu en dessous de la broche 13. Elles vont s'allumer au rythme de l'envoi (TX, Transmit) ou de la réception (RX, Receive) de bits de données sur la prise USB.

Des broches españoles ?

Plusieurs broches présentent un petit symbole ~ (un tilde) avant leur numéro. Il s'agit des broches 3, 5, 6, 9, 10 et 11. Ces six broches peuvent être utilisées d'une manière un peu spéciale. À côté de la mention DIGITAL, tu vois la mention PWM. C'est une technique pour utiliser une sortie numérique de manière à simuler une sortie analogique. Nous verrons cela dans la Semaine 2.

Analogique ou numérique ?

Quand tu allumes ou éteins la lumière de ta chambre, tu utilises un mécanisme numérique, car la lumière est soit totalement allumée, soit totalement éteinte. En revanche, au long d'une journée, le soleil diffuse une lumière qui varie graduellement : c'est un phénomène analogique. Il ne disparaît pas en un éclair à 18 heures pile.

Quand tu règles le volume de ta chaîne stéréo, la variation est continue. C'est un phénomène analogique également. En revanche, ton grille-pain est soit en train de dorer une tartine, soit il l'éjecte. Il ne reste pas à mi-hauteur (sauf bien sûr si une tartine se coince). C'est un phénomène numérique, discontinu plus précisément.

Un ordinateur ne connaît à l'intérieur que des choses discontinues, numériques : il n'y a que des 0 et des 1. Soit il y a du courant, soit il n'y en a pas. Pourtant, dans le monde physique (IRL), il y a des composants qui doivent être contrôlés de façon analogique, par exemple le volume sonore que l'on envoie à un haut-parleur. C'est pour pouvoir contrôler graduellement de tels composants que ton mikon possède certaines broches de sortie numériques pouvant être utilisées selon le principe PWM dont j'expliquerai les détails en temps utile.

Ton mikon peut donc parler en changeant l'état des broches numériques de sortie. C'est un langage vraiment élémentaire puisqu'il n'est constitué que de 0 et de 1, mais n'oublie pas qu'il peut changer l'état de chacune des broches plusieurs milliers de fois par seconde. Si tu pouvais envoyer du code Morse à cette vitesse, un livre entier pourrait être envoyé en quelques secondes. Ce n'est donc pas si ridicule que cela.

Renvoyer des résultats vers le monde extérieur pour contrôler des composants est une chose, mais il serait formidable de pouvoir écouter ce qui se passe dans le monde. C'est à cela que servent les entrées numériques.

Les entrées numériques

Inutile d'aller chercher bien loin : les entrées numériques se font sur les mêmes broches que les sorties numériques. En effet, dans tes programmes, tu peux à tout moment choisir d'utiliser une broche parmi les quatorze comme entrée ou comme sortie.

Lorsqu'elles sont utilisées en entrées numériques, les six broches PWM sont strictement identiques aux autres. En revanche, ma remarque concernant les deux broches 0 et 1 reste valable. Si tu utilises le Moniteur série dans l'atelier comme on va le découvrir

dans le prochain chapitre, n'utilise pas ces broches dans tes montages.

Je viens de dire qu'il s'agissait de broches d'entrées numériques. Autrement dit, elles peuvent seulement détecter qu'il y a 0 V ou 5 V ; mais que se passe-t-il avec 4 V sur une entrée ?

Le fabricant de ton mikon, le fondeur, qui est la société Atmel, a choisi de créer une zone de sécurité entre les deux états logiques bas (0 V) et haut (5 V) :

- Entre 0 V et 0,8 V, c'est un état bas, donc un 0.
- Entre 2,8 V et 5 V, c'est un état haut, donc un 1.
- Entre 0,8 V et 2,8 V, l'état est indéterminé, et le dernier état reste valable.

Ces broches d'entrées numériques vont te servir à détecter toutes sortes de changements, par exemple le fait d'appuyer sur un petit bouton-poussoir peut faire basculer une entrée de 0 V à 5 V ou de 5 V à 0 V. Cela te permet de détecter que quelqu'un a pressé le bouton. Dans un système d'alarme, le fait qu'une fenêtre s'ouvre va changer l'état d'un contacteur et tu pourras le détecter sur une broche d'entrée numérique.

Mais comment faire pour récupérer une valeur analogique comme une température ou une intensité de lumière naturelle ? Ce sont des valeurs analogiques, pas numériques. Dans ce cas, il faudrait disposer d'entrées analogiques. Justement, la carte Arduino en offre six.

Les entrées analogiques

Revenons maintenant aux entrées analogiques en partie basse de la carte (Figure 1.7). Le connecteur de droite porte la légende `ANALOG IN`. Bingo ! C'est ce qui nous intéresse maintenant.

Pour ces six entrées analogiques, la légende va de A0 à A5 (ici aussi, on commence à compter à 0).

C’est sur ces broches que tu vas pouvoir faire venir un signal qui varie entre 0 et 5 V de façon progressive. Par exemple, si tu veux gérer un bouton de réglage de volume, et que ce bouton envoi 5 V lorsqu’il est à fond, le fait de mettre le bouton à mi-chemin doit présenter 2,5 V sur l’entrée analogique sur laquelle tu l’as branché. Mais il y a quelque chose d’étrange, non ?

J’ai dit que ton mikon ne savait comprendre que des bits (des 0 et des 1). Comment pourrait-il distinguer entre une tension de 2,5 V et une tension de 2,6 V ? En plus, ces deux tensions sont dans la plage grise. Ce n’est ni un 0, ni un 1. Il doit y avoir une astuce.

La question est celle-ci : peut-on lire une valeur analogique avec une machine purement numérique telle qu’un microcontrôleur ? Ma réponse est simple : Yes you CAN. En effet, c’est un circuit appelé CAN, Convertisseur Analogique-Numérique (en anglais DAC, Digital Analog Converter) qui fait le traducteur. Nous utiliserons un tel convertisseur dans la prochaine partie.

Un cerveau vide à la naissance ?

Quand tu déballes ta carte Arduino Uno, elle est neuve, personne n’a encore travaillé avec (enfin, sauf si tu l’as achetée d’occasion). Puisqu’elle est neuve, elle ne devrait encore rien savoir faire puisque tu n’y as encore installé aucun programme…

En fait, le fabricant de la carte a eu la bonne idée, une fois la carte construite et testée, d’y installer un petit programme minimal qui permet dès la mise sous tension de vérifier qu’elle fonctionne. Ce programme est l’un des plus simples programmes fournis dans la série d’exemples que tu vas récupérer en même temps que l’atelier de développement IDE Arduino.

Puisque ce petit programme est déjà en place, nous allons vérifier que la carte fonctionne bien.

Séquence pratique : testons notre

bébé

Pour ce premier test, il te faut réunir :

- ta carte Arduino Uno ;
- son câble USB.

Et c'est tout !

1. Commence par organiser ton environnement pour que tu puisses poser la carte Arduino sur ton plan de travail sans qu'il y ait un risque de court-circuit avec les contacts du dessous de la carte.

   En effet, il faut que tu puisses voir le côté de la carte où se trouvent les connecteurs et les composants. De ce fait, les contacts du dessous pourraient toucher un objet métallique qui traînerait par là.

2. Branche d'abord un bout du câble USB dans une prise USB de l'ordinateur.

3. Tiens fermement la carte Arduino avec ta main principale (la droite pour la plupart des gens), puis insère le connecteur USB dans la prise du côté gauche de la carte Arduino.

   Ton mikon se réveille ! Continue à maintenir la carte si nécessaire car les boucles du câble peuvent emporter la carte et la faire tomber, car elle est légère.

Pour en savoir plus sur les circuits électroniques, je te conseille, dans la même collection, *Découvrir l'électronique en s'amusant*.

4. Maintenant, observe bien les petites lumières (des LED) qui se trouvent sur la carte. Repère celle portant la légende **LED** ou **L**. Elle doit s'allumer et s'éteindre à rythme régulier.

   Une autre LED marquée **ON** reste allumée en permanence, c'est celle qui confirme que ta carte est sous tension. Mais revenons à celle qui clignote. Pourquoi clignote-telle ? Il y a deux possibilités :

   - Soit elle clignote parce qu'il y a sur la carte un circuit que l'on appelle *oscillateur* (à base de condensateurs et de résistances). Ce n'est pas le cas, car cela ne donnerait pas exactement le même résultat. On verrait la LED devenir de plus en plus, puis de moins en moins brillante au fur et à mesure que le condensateur se charge et se décharge. Comme ce livre n'est pas consacré à l'électronique, je n'en parle pas plus ici.
   - L'autre cause du clignotement de la LED est celle qui nous intéresse : ce sont des instructions exécutées par le mikon qui modifient périodiquement l'état d'une broche de sortie du microcontrôleur, soit 5 volts pour allumer la LED, soit 0 volt pour l'éteindre.

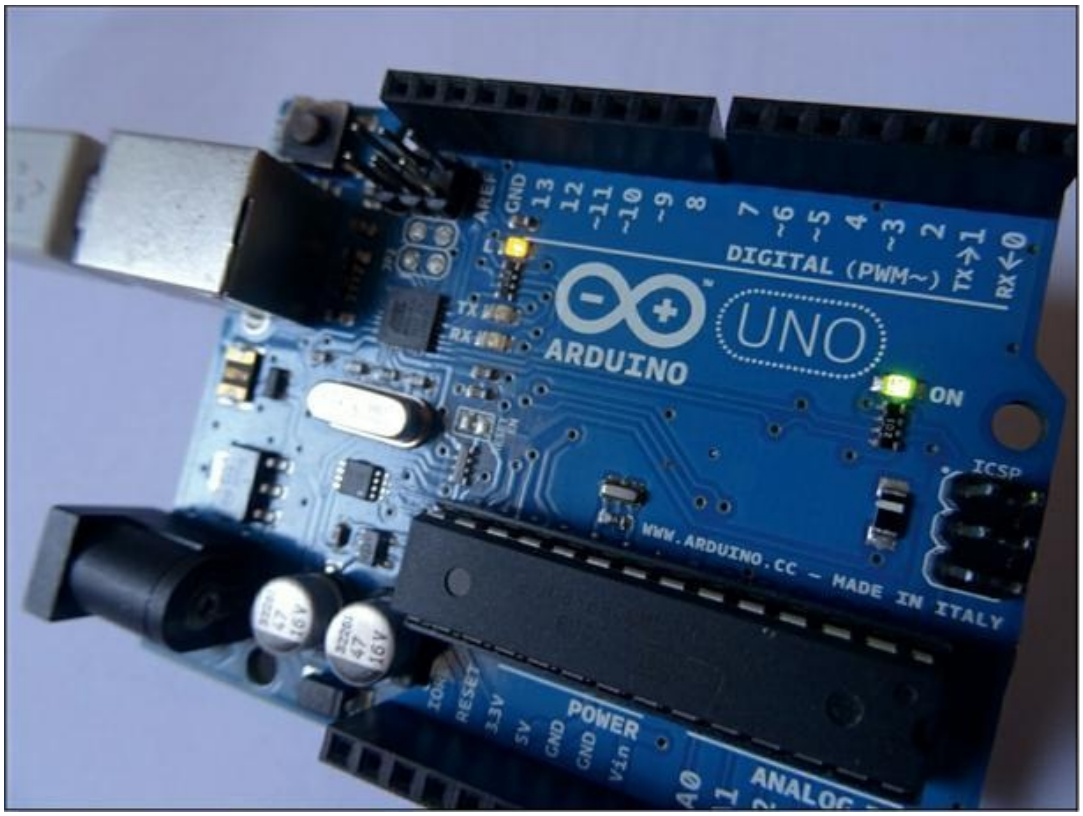


Figure 1.9 : Premier essai de la carte Arduino.

C'est exactement ce qui se passe. C'est un véritable programme qui est en train de fonctionner quand tu vois clignoter la LED ! Sans même avoir vu comment était écrit ce programme, essayons de deviner à quoi il devrait ressembler :

1. On commence avec une instruction pour forcer à 5 V la sortie du microcontrôleur sur laquelle est branchée cette petite LED.
2. Avec une instruction, on maintient la sortie dans le même état pendant un certain temps, à vue de nez, une demi-seconde.
3. On passe à une troisième instruction pour ramener l'état de la broche de sortie de la LED à 0 volt.

4. On enchaîne avec une quatrième instruction pour rester en pause dans le même état électrique.
5. Avec une dernière instruction, on force le mikon à revenir exécuter la première des quatre instructions (c'est un saut, JUMP !)

C'est exactement ce que fait le programme que nous allons découvrir bientôt, quand nous aurons installé le logiciel sur ton ordinateur pour afficher et modifier le code source des programmes.

Mais comment se fait-il qu'il y ait un programme dans la carte Arduino alors qu'elle était hors tension dans son emballage ? C'est tout simplement parce que les programmes sont écrits dans une mémoire Flash, qui ne s'efface pas quand on coupe l'alimentation. C'est le même genre de circuit mémoire que celui de tes clés USB.

Tu peux maintenant débrancher la carte pour qu'elle se repose un peu.

Langages de haut et de bas niveau

Quand tu vas programmer ton Arduino, tu vas utiliser un langage de programmation (un infolangage). Celui le plus utilisé sur un Arduino est le langage C, très répandu dans le monde.

Chaque ligne que tu vas écrire dans ce langage peut représenter des dizaines ou des centaines d'instructions machine du mikon. Voici quelques lignes en langage C :

```
if (heure == 7) {
  sHabiller();
  sortir();
}
```

C’est loin d’être incompréhensible, non ? S’il est 7 heures, on s’habille et on sort.

Un programme spécial (le compilateur) analyse une à une les lignes de ton texte source pour produire un autre fichier contenant une série d’instructions machine.

Voici quelques lignes en langage machine (Figure 1.10) telles que visualisées dans un programme d’édition hexadécimale. Chaque couple de chiffres représente une série de huit bits à 0 ou à 1, ce qui rend la lecture un (tout) petit peu plus facile.

C’est ce fichier qui sera envoyé dans la mémoire Flash du mikon par l’opération de téléversement. Dès que ce transfert est terminé, le programme démarre.

Il est possible d’écrire dans un langage quasiment identique aux instructions machine et c’est le langage de bas niveau appelé assembleur. Laissons-le de côté dans ce livre ; nous avons bien d’autres choses à découvrir et à expérimenter d’abord.

Quand tu auras terminé ce livre, tu pourras, avec précaution, aller voir là où se tapissent les fichiers produits par le compilateur. Par exemple, sous Windows : `C:\Users\ton_nom\AppData\Local\Temp\arduino_build_999999`

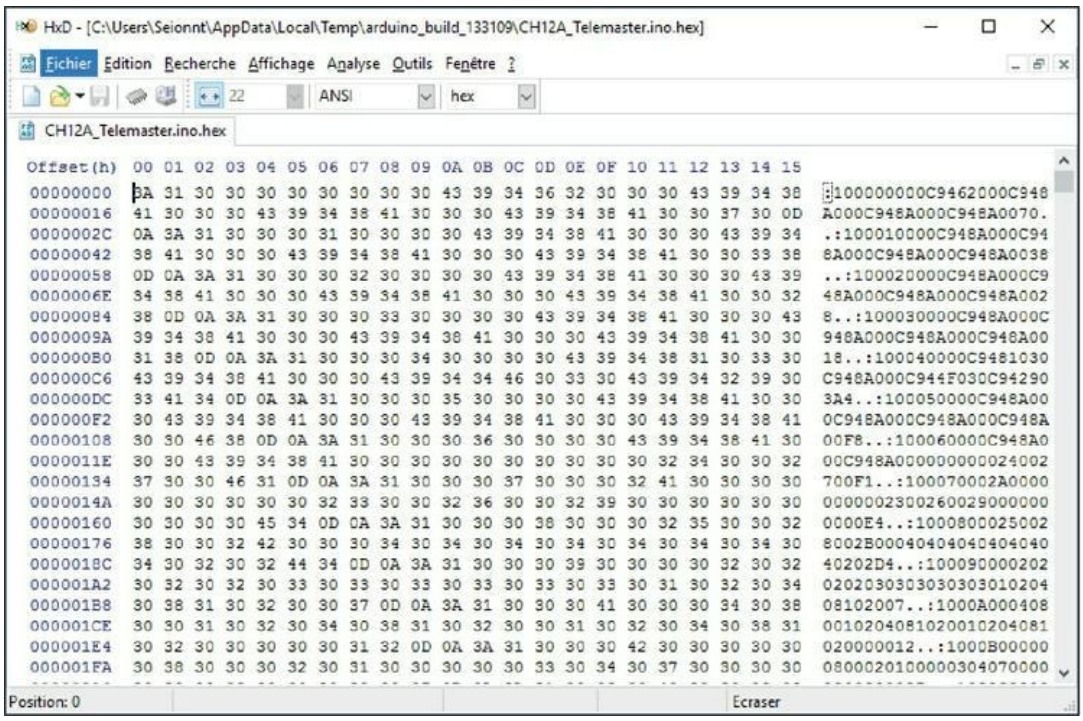


Figure 1.10 : Aspect de ce qui est téléversé vers le mikon.

Récapitulons

Nous allons maintenant découvrir les outils avec lesquels tu vas pouvoir écrire le texte source de tes programmes, puis le faire traduire et le transférer vers le mikon sur la carte.

Je te rappelle que j'ai listé dans l'introduction du livre les composants à se procurer pour réaliser les projets.

Chapitre 2
L'atelier du cyberartisan

AU MENU DU CHAPITRE :

- **Un atelier, c'est mieux pour travailler et s'amuser**
- **C'est quand compile ?**

À la différence d'un ordinateur de bureau, tu ne peux mettre en place qu'un seul programme à la fois sur le mikon. Pour l'instant, il contient le programme qui a été installé en usine, celui qui fait clignoter une fois par seconde la mini-LED soudée, reliée à la broche 13.

Pour pouvoir y installer un autre programme, il faut utiliser un outil qui va téléverser un fichier binaire qui aura d'abord été créé par un autre outil, le compilateur. Mais avant cela, il faut avoir écrit le texte source, avec un outil qui ressemble un peu à un traitement de texte ou au Bloc-notes de Windows.

Voyons donc d'abord comment récupérer l'excellent logiciel gratuit pour créer tes programmes. Il réunit plusieurs outils :

- un éditeur de texte pour rédiger et modifier le code source ;
- un compilateur qui va traduire ce texte en langage binaire ;
- un téléverseur qui va envoyer le fichier binaire vers la mémoire Flash du mikon.

Installation de l'atelier Arduino IDE

L'outil unique avec lequel tu vas faire tes premiers pas (et les suivants) est un atelier de développement logiciel. On dit également IDE, qui vient de l'anglais Integrated Developement Environment. Nous allons d'abord récupérer ce programme, l'installer puis le paramétrer pour qu'il fonctionne avec ta carte.

1. Dans ton navigateur préféré, recherche les deux mots suivants :

   `Arduino IDE`

   Pourquoi ne pas en profiter pour découvrir le moteur européen Qwant (`www.qwant.com`) ?

   Dans les réponses fournies par le moteur de recherche, choisis celle qui correspond à l'adresse suivante :

   `https://www.arduino.cc/en/Main/Software`

   Tu devrais voir apparaître une page dans le style de celle illustrée en Figure 2.1.

Arduino - Software - Mozilla Firefox
Fichier Édition Affichage Historique Marque-pages Outils ?
Arduino IDE - Quant Rech...
Arduino - Software
https://www.arduino.cc/en/Main/Software
Rechercher
Les plus visités
Débuter avec Firefox
ARDUINO
Search the Arduino Website
Home
Buy
Software
Products
Learning
Forum
Support
Blog
LOG IN
SIGN UP
SOFTWARE
ENGLISH
Access the Online IDE
ARDUINO WEB EDITOR
Start coding online with the Arduino Web Editor, save your sketches in the cloud, and always have the most up-to-date version of the IDE, including all the contributed libraries and support for new Arduino boards. The Arduino Web Editor is one of the Arduino Create platform's tools.
Try It Now
Getting Started

Figure 2.1 : La page de téléchargement de l'atelier IDE.

Cet outil existe dorénavant en version Web (**Access the Online IDE**), mais ce n'est pas celle qui nous intéresse. Descends un peu dans la page jusqu'à voir apparaître la zone intitulée **Download the Arduino IDE**.

Buy
Software
Products
Learning
Forum
Support
Blog
LOG IN
SIGN UP
Download the Arduino IDE
ARDUINO 1.8.1
The open-source Arduino Software (IDE) makes it easy to write code and upload it to the board. It runs on Windows, Mac OS X, and Linux. The environment is written in Java and based on Processing and other open-source software.
This software can be used with any Arduino board.
Refer to the Getting Started page for installation instructions.
Windows Installer
Windows ZIP file for non admin install
Windows app Get
Mac OS X 10.7 Lion or newer
Linux 32 bits
Linux 64 bits
Linux ARM
Release Notes
Source Code
Checksums (sha512)
ARDUINO SOFTWARE
HOURLY BUILDS
LAST UPDATE
20 February 2017 19:11:41 GMT
ARDUINO 1.0.6 / 1.5.x / 1.6.x
PREVIOUS RELEASES

Figure 2.2 : Section de téléchargement dans la page Arduino.

Dans la partie droite, tu as accès aux liens des différentes versions de l'atelier, selon le système d'exploitation que tu utilises.

Atelier pour Windows

1. Dans la section **Download** présentée dans la Figure 2.2, clique **Windows Installer** ou **Mac OS X**.
2. La page qui apparaît te propose de contribuer au financement du mouvement Arduino. Pour l'instant, tu vas découvrir quel est ce mouvement.
3. À gauche du bouton **Contribute and Download**, clique le lien **Just Download**, moins visible (Figure 2.3).

Buy
Software
Products
Learning
Forum
Support
Blog
LOG IN
SIGN UP
Support the Arduino Software
Consider supporting the Arduino Software by contributing to its development. (US tax payers, please note this contribution is not tax deductible). Learn more on how your contribution will be used.
SINCE MARCH 2015, THE ARDUINO IDE HAS BEEN DOWNLOADED 13,660,013 TIMES. (IMPRESSIVE!) NO LONGER JUST FOR ARDUINO AND GENUINO BOARDS, HUNDREDS OF COMPANIES AROUND THE WORLD ARE USING THE IDE TO PROGRAM THEIR DEVICES, INCLUDING COMPATIBLES, CLONES, AND EVEN COUNTERFEITS. HELP ACCELERATE ITS DEVELOPMENT WITH A SMALL CONTRIBUTION! REMEMBER: OPEN SOURCE IS LOVE!
$3
$5
$10
$25
$50
OTHER
JUST DOWNLOAD
CONTRIBUTE & DOWNLOAD

Figure 2.3 : Le bouton de téléchargement simple.

En fonction des réglages de ton navigateur, soit le téléchargement démarre directement pour copier le fichier dans le dossier `Téléchargements`, soit il te demande si tu veux enregistrer le fichier et à quel endroit (Figure 2.4).

Ouverture de arduino-1.8.1-windows.exe
Vous avez choisi d'ouvrir :
arduino-1.8.1-windows.exe
qui est un fichier de type : Binary File (89,4 Mo)
à partir de : https://downloads.arduino.cc
Voulez-vous enregistrer ce fichier ?
Enregistrer le fichier
Annuler

Figure 2.4 : Choix d'enregistrement du fichier.

Dans ce deuxième cas, choisis l'endroit où tu vas télécharger le fichier, par exemple dans `Téléchargements`. Tu remarques que le fichier

commence par le nom Arduino, suivi du numéro de version. À l'heure où j'écris ces lignes, nous en sommes à la version 1.8.

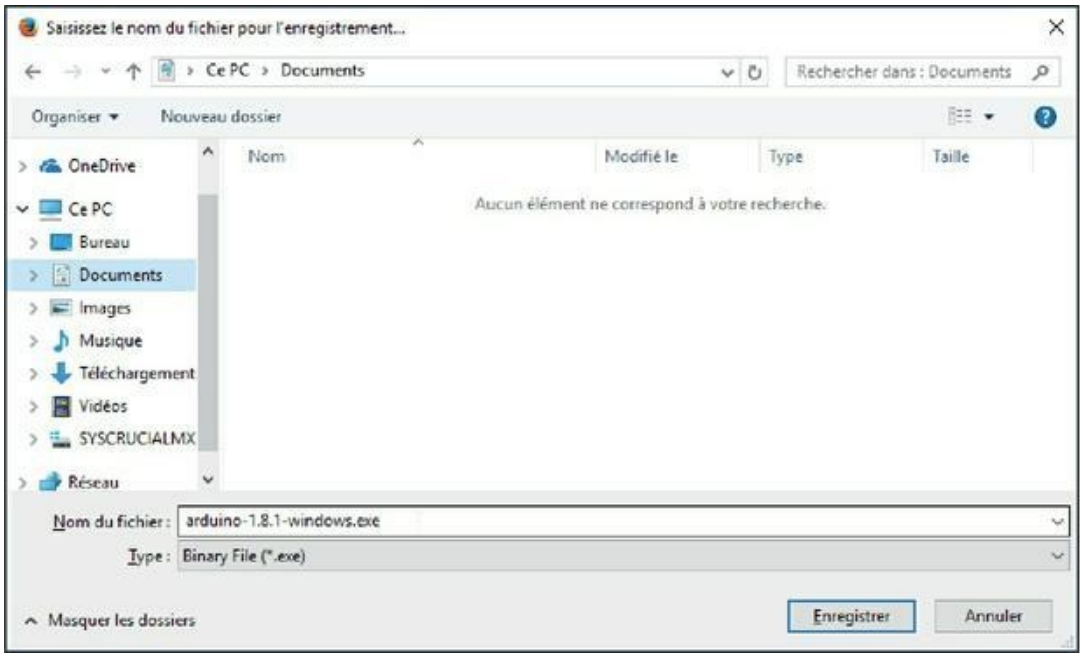


Figure 2.5 : Choix du dossier de stockage.

4. Ouvre si nécessaire ton Explorateur, Finder ou Navigateur de fichiers et rends-toi dans le dossier dans lequel tu as demandé de déposer le fichier téléchargé. Tu dois voir apparaître un fichier portant le nom « Arduino-numéro de version ».

5. Double-clique le nom du fichier pour démarrer l'installation.

Sous Linux

Sous Linux, le plus simple consiste à utiliser ton gestionnaire de paquetages en cherchant Arduino. Toute la suite va s'enchaîner

automatiquement.

Installation de l'atelier

Pour garantir que l'installation se fera avec tous les droits d'accès, clique du bouton droit dans le nom ou dans l'icône de l'installateur et choisis **Exécuter en tant qu'administrateur** (cette précaution n'est pas toujours nécessaire).

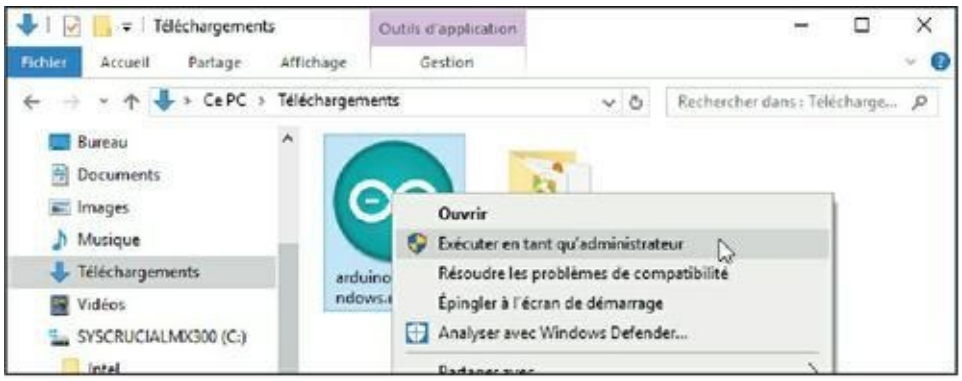


Figure 2.6 : Démarrage de l'installateur sous Windows.

1. Si une boîte message standard te demande de confirmer que tu autorises l'application à modifier ton ordinateur, clique Oui.

   La première boîte de dialogue de l'installateur apparaît.

2. Il suffit de cliquer en bas à droite le bouton **I Agree**. Cela confirme que tu acceptes d'utiliser le logiciel dans les conditions de la Free Software Foundation. C'est la fondation au niveau mondial qui gère les droits des auteurs de logiciels open source.

Arduino Setup: License Agreement
Please review the license agreement before installing Arduino. If you accept all terms of the agreement, click I Agree.
GNU LESSER GENERAL PUBLIC LICENSE
Version 3, 29 June 2007
Copyright (C) 2007 Free Software Foundation, Inc. <http://fsf.org/>
Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.
This version of the GNU Lesser General Public License incorporates the terms and conditions of version 3 of the GNU General Public License, supplemented by the additional permissions listed below.
Cancel
Nullsoft Install System v3.0
I Agree

Figure 2.7 : Démarrage de l'installateur sous Windows.

3. Dans les options suivantes, il n'y a rien à changer. Tu vas installer le logiciel Arduino, le pilote USB, un raccourci dans le menu Démarrer ou dans Applications, un raccourci sur le Bureau, et tu vas créer tous les fichiers sources avec l'extension `.ino` qui dénote le format des fichiers sources Arduino. Tu peux à présent directement cliquer le bouton **Next** pour passer à l'étape suivante.

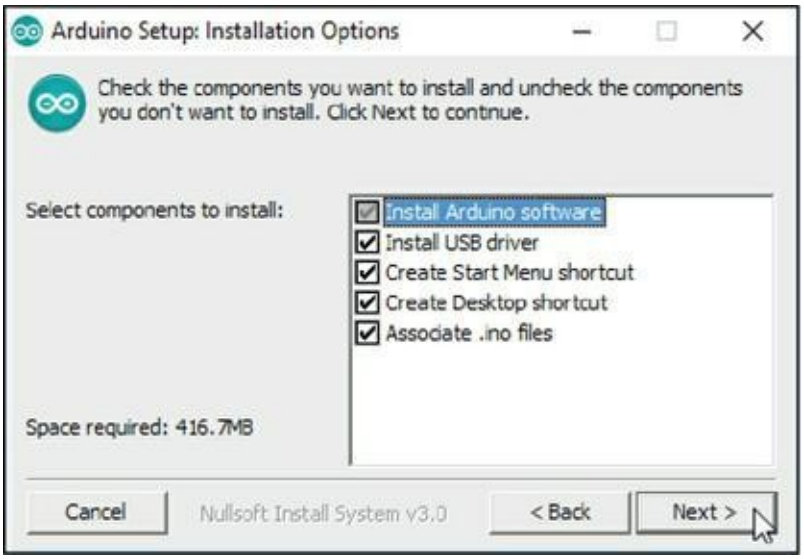

Arduino Setup: Installation Options
Check the components you want to install and uncheck the components you don't want to install. Click Next to continue.
Select components to install:
Install Arduino software
Install USB driver
Create Start Menu shortcut
Create Desktop shortcut
Associate .ino files
Space required: 416.7MB
Cancel
Nullsoft Install System v3.0
< Back
Next >

Figure 2.8 : Choix des composants à installer.

À la troisième étape, tu peux changer le dossier dans lequel va être installé l'atelier. Ne change rien à ce niveau sauf si tu sais ce que tu veux faire. Clique le bouton **Install** pour lancer le travail.

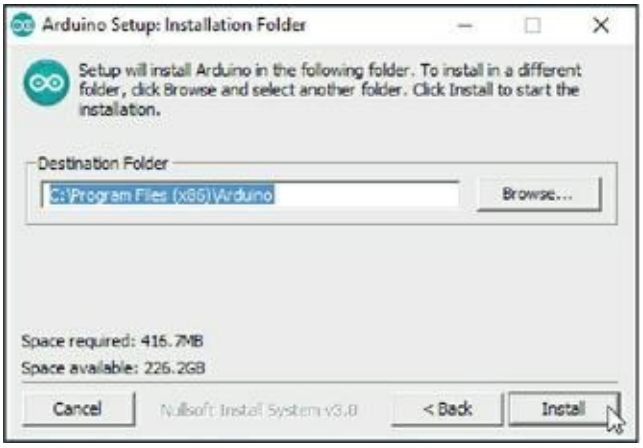

Arduino Setup: Installation Folder
Setup will install Arduino in the following folder. To install in a different folder, click Browse and select another folder. Click Install to start the installation.
Destination Folder
C:\Program Files (x86)\Arduino
Browse...
Space required: 416.7MB
Space available: 226.2GB
Cancel
Nullsoft Install System v3.0
< Back
Install

Figure 2.9 : Choix du dossier d'installation de l'atelier.

Il n'y a plus qu'à admirer le travail d'installation en cours (Figure 2.10). Tu peux même afficher les détails si cela t'intéresse.

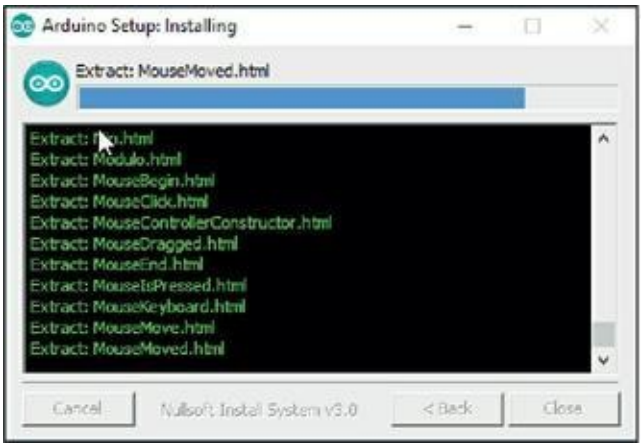


Figure 2.10 : Installation en cours.

Tu constates qu'il y a plus de 3 000 fichiers à installer. Tous ne se seront pas utiles, car l'installation couvre tous les modèles de cartes Arduino, et pas seulement la Uno.

Une fois l'installation terminée, il est possible qu'une boîte de sécurité surgisse pour te demander si tu autorises l'installation d'un pilote de périphérique (*driver*). Ce pilote est indispensable : il permet de communiquer avec la carte *via* le port USB.

9. Clique le bouton **Install** (Figure 2.11).

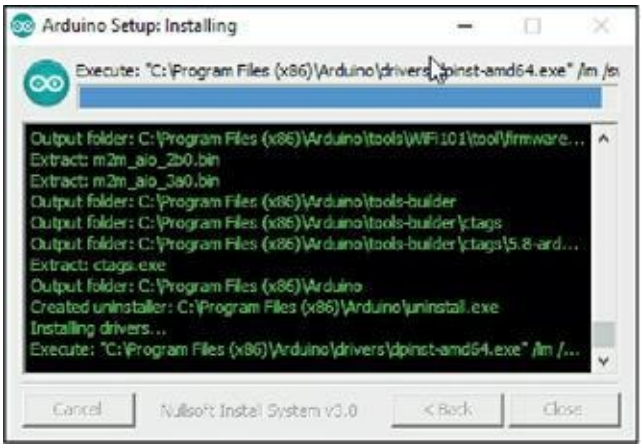


Figure 2.11 : Installation du pilote périphérique.

Il est possible qu'une deuxième et une troisième boîte de demande d'autorisation apparaissent ensuite. Il suffit de répondre **Install**.

Si le logiciel Pare-feu du système te demande d'autoriser l'installation de Java, réponds **Autoriser l'accès**.

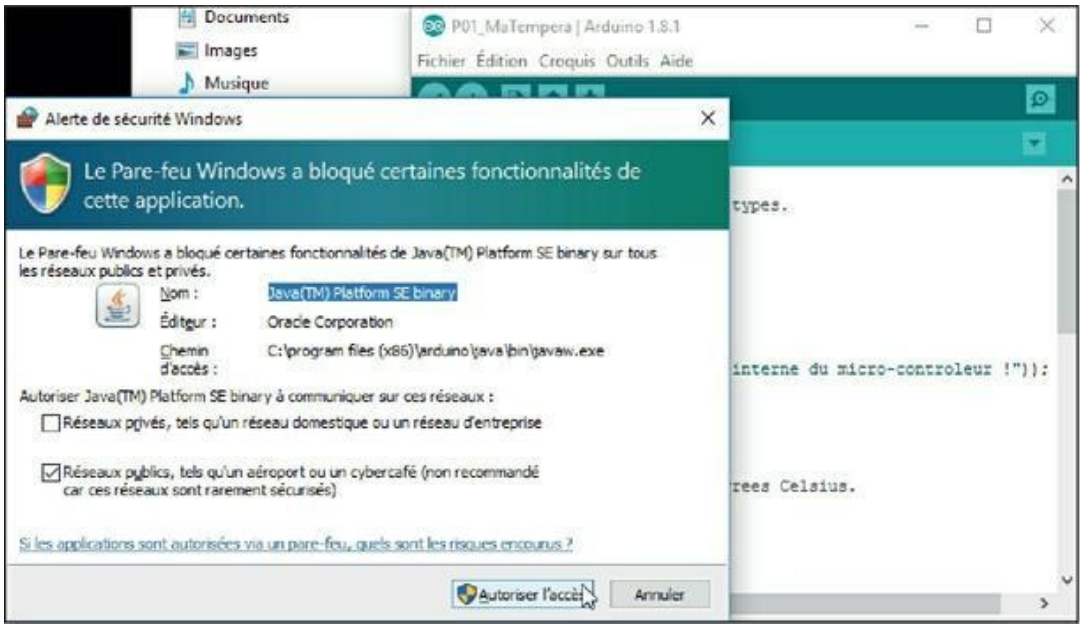

Documents
Images
Musique
P01_MaTempera | Arduino 1.8.1
Fichier Édition Croquis Outils Aide
Alerte de sécurité Windows
Le Pare-feu Windows a bloqué certaines fonctionnalités de cette application.
Le Pare-feu Windows a bloqué certaines fonctionnalités de Java(TM) Platform SE binary sur tous les réseaux publics et privés.
Nom : Java(TM) Platform SE binary
Éditeur : Oracle Corporation
Chemin d'accès : C:\program files (x86)\arduino\java\bin\javaw.exe
Autoriser Java(TM) Platform SE binary à communiquer sur ces réseaux :
Réseaux privés, tels qu'un réseau domestique ou un réseau d'entreprise
Réseaux publics, tels qu'un aéroport ou un cybercafé (non recommandé car ces réseaux sont rarement sécurisés)
Si les applications sont autorisées via un pare-feu, quels sont les risques encourus ?
Autoriser l'accès
Annuler
types.
interne du micro-controleur !"));
rees Celsius.

Figure 2.12 : Installation du pilote périphérique.

10. Tu reviens à la boîte principale d'installation. Le titre indique **Completed** . Clique le bouton **Close** : l'installation est terminée.

Voyons maintenant comment configurer l'atelier.

Premier démarrage de l'atelier

IDE

Maintenant que tu as installé le logiciel atelier, tu as sans doute envie de l'essayer au plus vite. Alors allons-y :

Procédure pratique

1. Si la carte n'est pas branchée sur l'ordinateur, rebranche-la, en faisant attention à la poser sur une surface non métallique.
2. Cherche sur le bureau ou dans les menus le nom Arduino et double-clique pour lancer le programme.
3. Tu vois apparaître la fenêtre principale de ton atelier (Figure 2.13). Nous en ouvrirons une autre de temps en temps, mais en général, ce n'est que dans cette fenêtre que nous allons travailler. Qu'est-ce que tu découvres ?

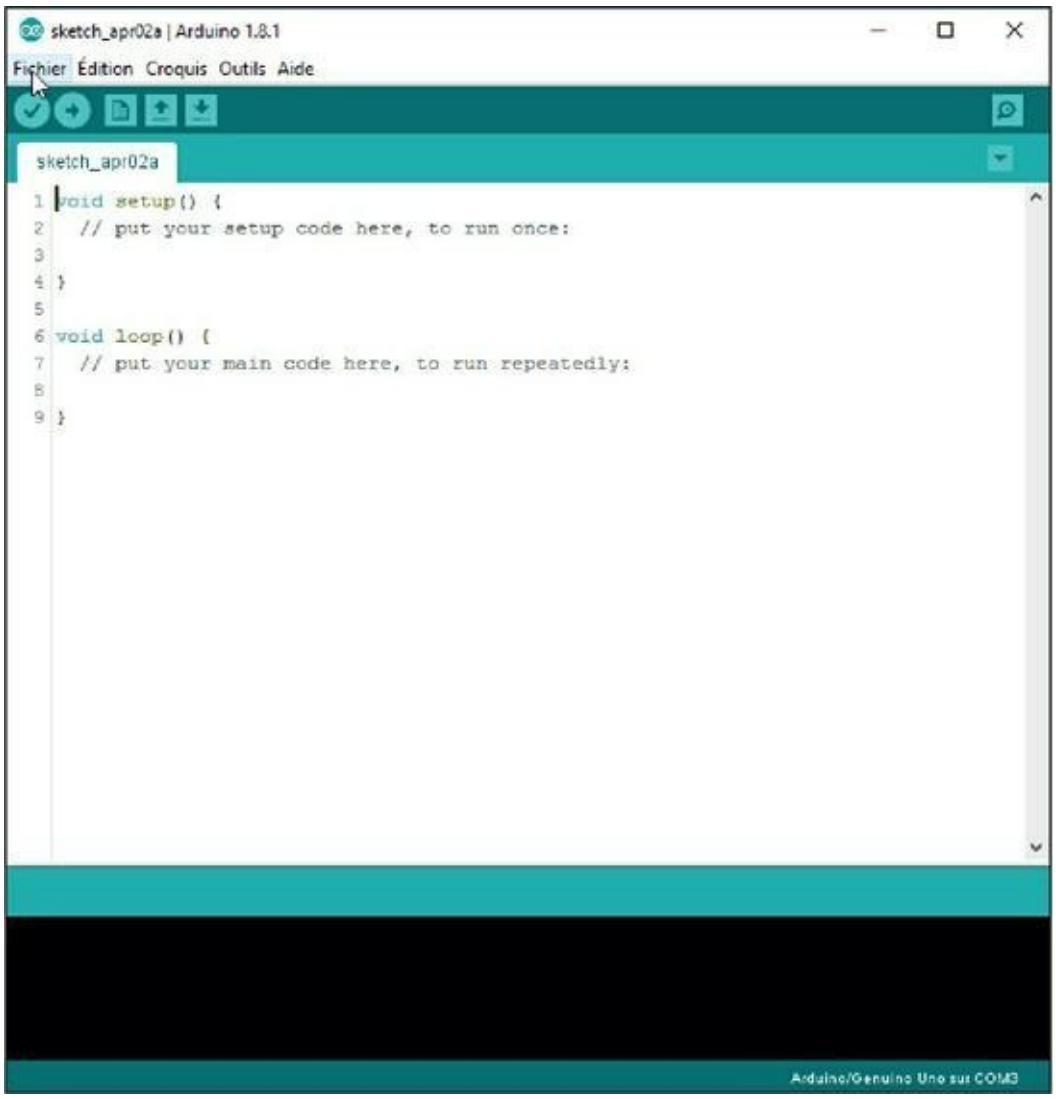

sketch_apr02a | Arduino 1.8.1
Fichier Édition Croquis Outils Aide
sketch_apr02a
void setup() {
// put your setup code here, to run once:
}
void loop() {
// put your main code here, to run repeatedly:
}
Arduino/Genuino Uno sur COM3

Figure 2.13 : Vue générale de l'atelier Arduino.

Tout en haut, la barre de titre avec une mention étrange juste avant le nom du programme et le numéro de version :

```
sketch_apr02a
```

Le mot sketch se traduit par « croquis » . C'est ainsi que les inventeurs de l'Arduino ont décidé d'appeler les programmes, c'est-à-dire le code source que tu écris. Dès que l'atelier démarre, il ouvre un croquis pour pouvoir démarrer immédiatement. Ce croquis n'est pas entièrement vide, comme tu peux le constater. Nous y reviendrons plus tard.

La mention `apr02a` signifie qu'il s'agit du premier nouveau projet du 2 avril.

On retrouve ensuite classiquement une barre de menus, avec **Fichier**, **Édition**, **Croquis**, **Outils** et **Aide**. Nous en découvrirons les détails au fur et à mesure des besoins.

Sous la barre de menus, il y a une barre de boutons que nous apprendrons à utiliser au fur et à mesure.

Vient ensuite la grande zone blanche dans laquelle tu écris le code source de tes programmes, comme dans un traitement de texte. Elle est presque vide pour l'instant. Nous reviendrons plus loin sur cet embryon de programme.

L'onglet en haut rappelle le nom du programme en cours.

Dans la partie inférieure, sous un bandeau vert, nous trouvons un grand panneau noir. C'est ici que s'afficheront les messages de la part de l'atelier, notamment lorsqu'il va détecter une erreur dans ton programme.

Enfin, tout en bas, la barre d'état rappelle le modèle de carte détecté et le port de communication.

Justement, il nous faut passer par une petite étape de préparation qui ne sera réalisée qu'une fois.

Configuration de la carte

Pour établir le dialogue entre la carte et l'atelier, il faut définir deux paramètres : le type de carte et le nom du port de communication USB.

1. Ouvre le menu **Outils** puis le sous-menu **Type de carte**.
2. Tu découvres un très grand nombre de cartes : il est possible que le modèle **Arduino/ Genuino Uno** soit

déjà sélectionné. Si ce n'est pas le cas, sélectionne ce modèle de carte (Figure 2.14).

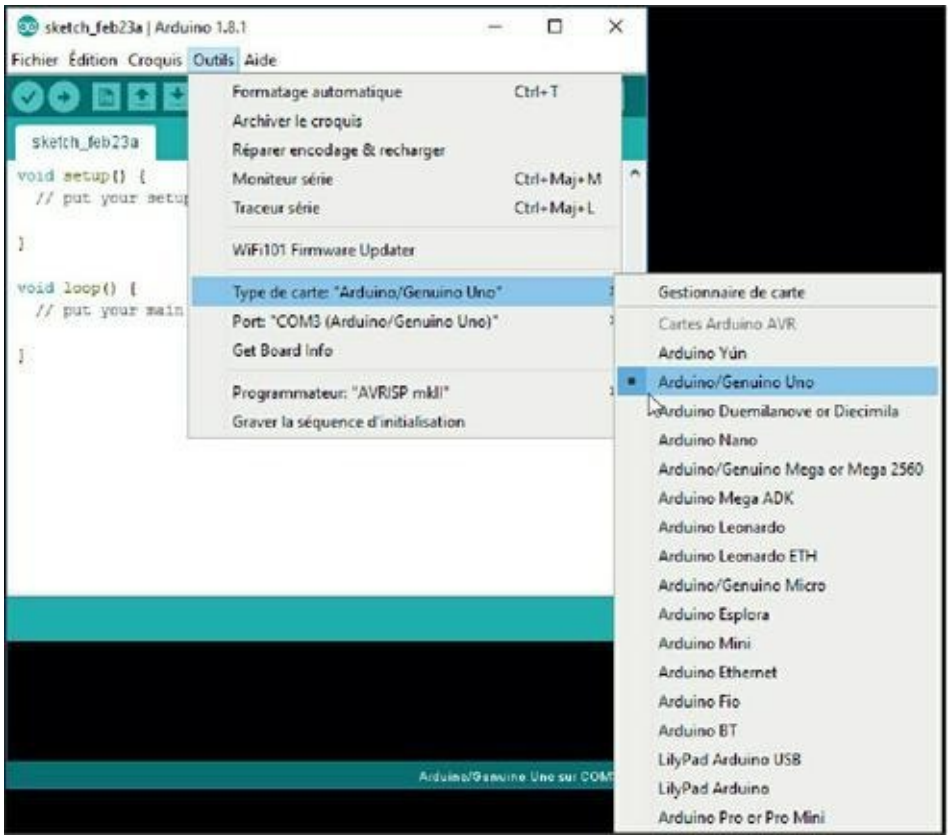


Figure 2.14 : Choix du modèle de carte.

3. Ouvre à nouveau le menu **Outils** et choisis le sous-menu **Port**(Figure 2.15). Ce menu est grisé si tu n'as pas branché la carte ou si elle n'est pas du tout détectée.

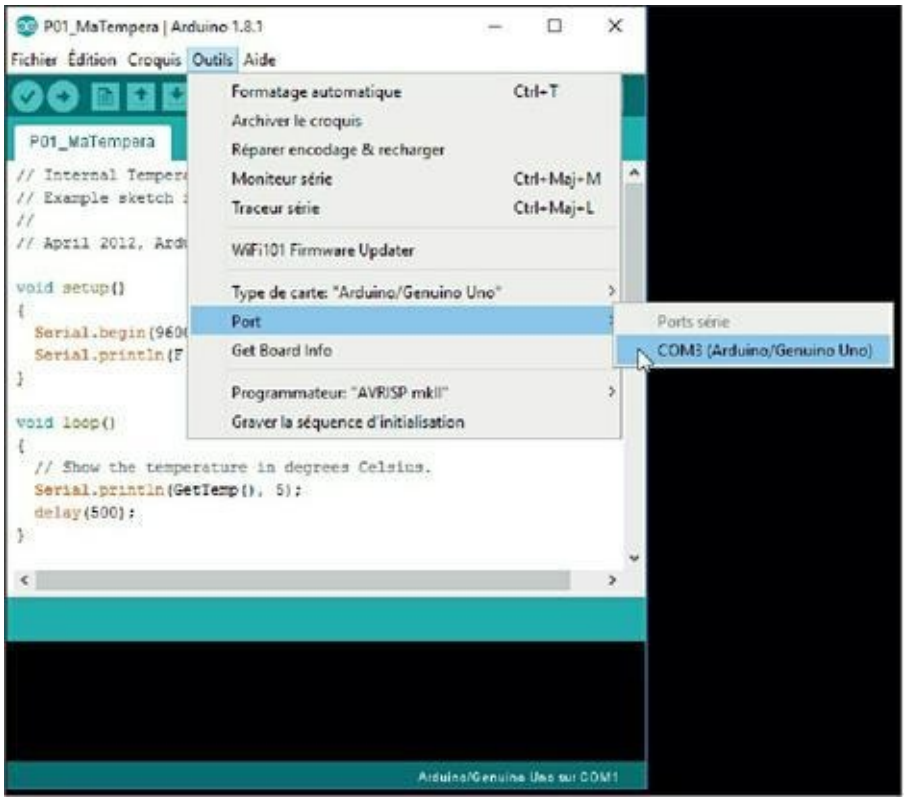


Figure 2.15 : Choix du port de communication.

Si la carte n'est pas détectée alors qu'elle est bien branchée par le câble USB et que le témoin de mise sous tension est allumé sur la carte, c'est qu'il y a un problème au niveau du pilote. Dans ce cas, rare, je ne peux que t'inviter à te renseigner sur le forum Arduino français.

4. Pour vérifier que tout est bien configuré, ouvre une fois de plus le menu **Outils** et choisis la commande **Get Board Info** (Interroger la carte).

Figure 2.16 : Message confirmant que la carte est détectée.

Tu dois voir apparaître une petite boîte de dialogue indiquant quatre paramètres, et notamment le numéro de série, **SN**. Sache-le : chaque carte est identifiable parmi toutes celles qui sont utilisées, de Valparaiso à Séoul.

5. Clique OK pour refermer la boîte.

Tu es maintenant prêt à lancer ta première vérification et ta première compilation.

Compilation de test...

Je t'expliquais dans le premier chapitre que les programmes que tu écris doivent être traduits en code exécutable avant d'être transférés vers la carte.

Cette opération se nomme la compilation. Dans l'atelier, cela correspond au bouton **Vérifier**, le premier de la barre de boutons.

Clique le bouton **Vérifier** et observe bien le contenu du panneau inférieur.

sketch_feb23a | Arduino 1.8.1
Fichier Édition Croquis Outils Aide
sketch_feb23a
void setup() {
// put your setup code here, to run once:
}
void loop() {
// put your main code here, to run repeatedly:
}
Compilation terminée.
Le croquis utilise 444 octets (1%) de l'espace de stockage de programmes. Le maximum est d
Les variables globales utilisent 9 octets (0%) de mémoire dynamique, ce qui laisse 2039 oct
1
Arduino/Genuino Uno sur COM3

Figure 2.17 : Messages d'une compilation réussie.

Deux lignes apparaissent dans le panneau inférieur :

» La première ligne dit que le croquis utilise quelques centaines d'octets de l'espace de stockage de programmes, c'est-à-dire la mémoire Flash. Le message rappelle d'ailleurs que tu peux au maximum installer 32 256 octets de code binaire, soit un peu moins de 32 ko. C'est bien la valeur indiquée par le constructeur pour la mémoire Flash.

» La seconde ligne parle de variables globales. C'est un peu étrange, puisque ce programme minimal n'utilise aucune variable. Pourtant, nous occupons 9 octets de la mémoire des données, c'est-à-dire la mémoire appelée *SRAM*. Cela correspond bien à ce que le constructeur prétend : il vend le mikon avec 2 048 octets de mémoire pour les variables : 2 048 moins les 9 octets que nous occupons font bien 2 039, comme affiché.

Ces messages ne sont pas anodins. En effet, un mikon n'a pas beaucoup de mémoire et il faudra toujours surveiller ces messages car on remplit vite l'espace disponible.

Ce programme ne fait rien, mais il sera accepté par le mikon parce qu'il contient les deux éléments obligatoires dans tout croquis Arduino : les deux fonctions qui portent obligatoirement les noms `setup()` et `loop()`. Nous verrons cela plus tard. Pour l'instant, essayons de transférer le programme vers la carte Arduino puisqu'il peut être compilé.

… et téléversement de test

1. Clique le deuxième bouton de la barre de boutons, **Téléverser**. Rien ne se passe, sauf qu'il n'y a pas de message d'erreur. Pas de nouvelles, bonnes nouvelles. Provoquons-le un peu, notre mikon !

2. Dans l'éditeur, place le curseur au tout début de la ligne de la fonction `setup()` :

```
void setup() {
```

3. Avec la touche **Suppr**, supprime la lettre v du début du mot void (Figure 2.18).

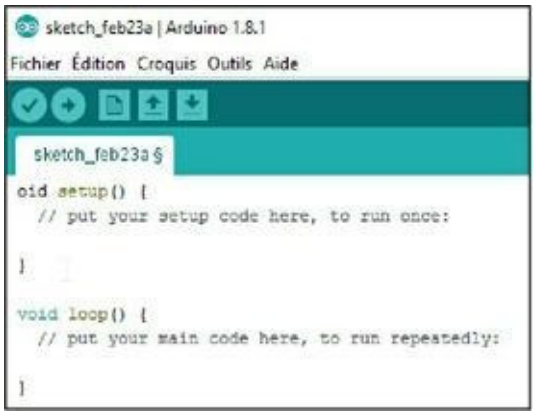

sketch_feb23a | Arduino 1.8.1
Fichier Édition Croquis Outils Aide
sketch_feb23a §
oid setup() {
// put your setup code here, to run once:
}
void loop() {
// put your main code here, to run repeatedly:
}

Figure 2.18 : Une erreur volontaire.

Dès que tu modifies le fichier vide proposé au démarrage, il faut enregistrer cette version modifiée dans un fichier pour pouvoir compiler le programme.

4. Dans le menu **Fichier**, choisis **Enregistrer sous**.

Tu vois apparaître une boîte standard (Enregistrer le dossier des croquis…). *A priori*, le fichier sera implanté dans un sous-dossier de `Mes documents`. Tu peux conserver cet emplacement pour y réunir tous les projets ou définir un nouveau dossier ailleurs si tu sais le faire.

5. Choisis avec soin le nom de ton dossier, car cela sera également le nom du fichier du projet. Par exemple :

`VideEtBogue`

6. Tu peux maintenant tenter une compilation avec le bouton **Vérifier** ou le raccourci clavier **Ctrl-R**.

Tu vois apparaître très vite deux lignes affichées en orange dans le panneau des messages, ce qui signifie que c'est un problème (Figure 2.19).

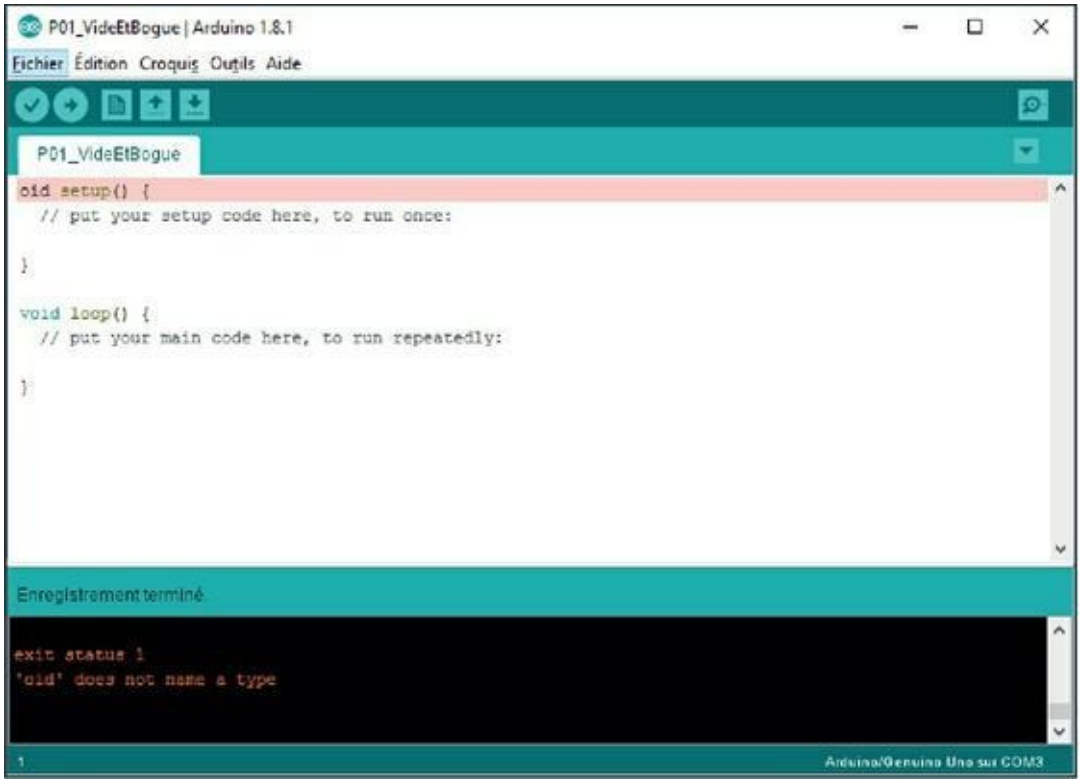


Figure 2.19 : Le compilateur n'est pas content.

Le message est en anglais, mais la deuxième ligne signifie que le mot oid ne désigne pas un type connu. Qu'est-ce que cela veut dire ?

C'est tout simplement que le compilateur doit normalement trouver à cet endroit un des mots réservés pour indiquer un type de données, c'est-à-dire la nature d'une donnée. Ici, il s'agit d'un type spécial, le type void qui signifie « vide » .

Tu as peut-être déjà feuilleté un livre dans une langue étrangère, par exemple en chinois, en hindi ou en arménien ? Regarde par exemple cet extrait :

Figure 2.20 : Une phrase en bengali.

Si tu ne connais pas la langue, tu ne sauras même pas prononcer les sons. Il te faut une indication.

Le compilateur du langage est beaucoup plus bête qu'un être humain. Il faut tout lui dire. Il faut donc notamment lui dire quand on indique un chiffre ou quand on indique une lettre. C'est à cela que sert le type de données. Ici, le type void signifie tout simplement qu'il n'y a pas de type. Nous y reviendrons. Contente-toi de savoir qu'il faut écrire sans fôte (sic) pour se faire obéir par le mikon.

Maintenant que tu as vu ton premier message d'erreur, remettons les choses en ordre :

1. Ajoute à nouveau la lettre v en minuscule au début du mot oid tout au début du programme.
2. Relance la compilation qui doit se dérouler sans problème.

Tu as maintenant terminé de mettre en place ton principal outil de travail, mais ce n'est pas le seul.

Terminons avec quelques outils qui vont te permettre de mieux t'amuser.

Autres outils

Très peu d'outils vont être nécessaires puisque ce livre est dédié à la programmation.

Soudure

Dans ce livre, j'ai choisi de ne pas utiliser de fer à souder. Toutefois, il en faudrait un lorsque le module t'est livré sans que les broches de connexion soient soudées.

Dans ce cas, deux solutions :

- Tu demandes autour de toi si quelqu'un peut faire les soudures.
- Si tu sais souder à l'étain, tu le fais toi-même, mais prends tes précautions.

Au pire, les rares exemples pour lesquels certains fournisseurs vendent le module non soudé resteront de côté le temps que tu puisses trouver la solution. Cela n'empêche pas de réaliser les autres projets.

Outils manuels

Les outils dont tu peux avoir besoin sont :

- Multimètre. Pour mesurer des volts, des ampères et des ohms.
- Pinces. Une petite pince à becs plats et une pince coupante.
- Tournevis. (option) Un mini-tournevis pour régler le potentiomètre de l'afficheur.
- Connexions. Une bonne réserve de fils de connexion que j'appelle *straps* (on dit aussi *jarretières*). Des mâle-mâle courts et longs, des mâle-femelle et quelques femelle-femelle.

- Plaques. Au moins une plaque d'essai ou de prototypage. Les trois tailles les plus utilisées sont :
  - la normale avec 800 trous environ, avec deux rails dans le sens vertical pour le négatif et le positif de l'alimentation ;
  - la demi-longueur avec 400 trous environ et les deux rails aussi sur les côtés ;
  - la miniature avec 170 trous, qui n'a pas de rails d'alimentation.

Avoir plusieurs plaques t'épargne de tout démonter pour passer à un autre projet.

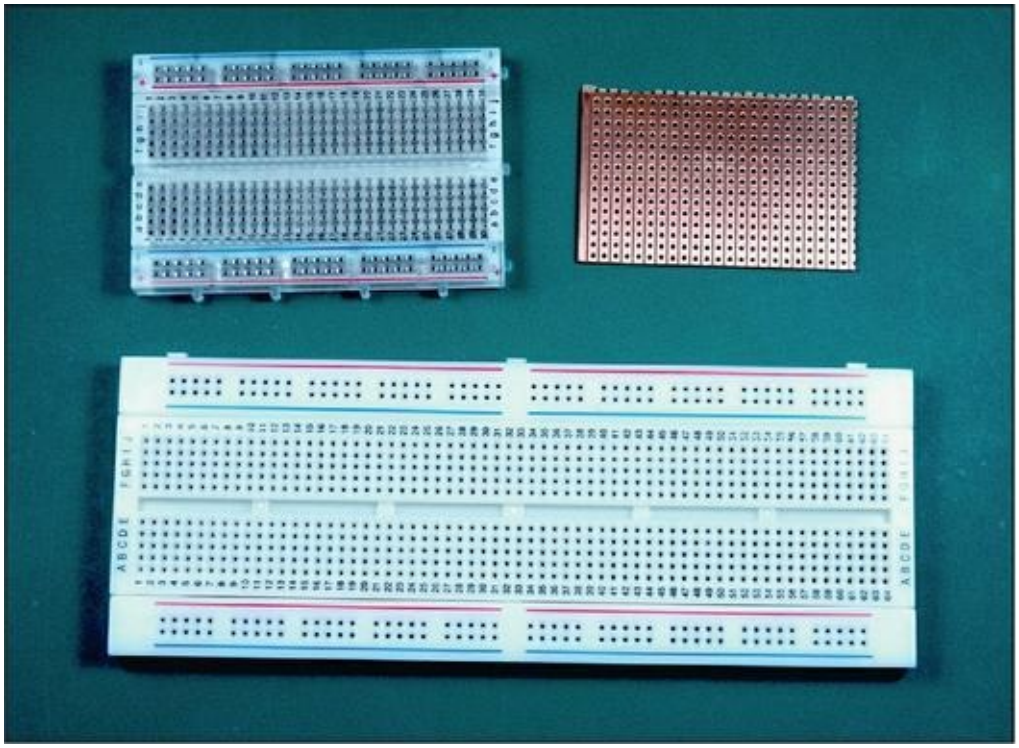


Figure 2.21 : Quelques plaques d'essai.

C’est grâce à ces plaques que tu n’as pas besoin de fer à souder. Les broches et pattes des composants s’enfichent dans les trous qui sont interconnectés 5 par 5 dans chaque demi-rangée. Dans la Figure 2.22, des traits jaunes montrent comment se font les contacts par-dessous.

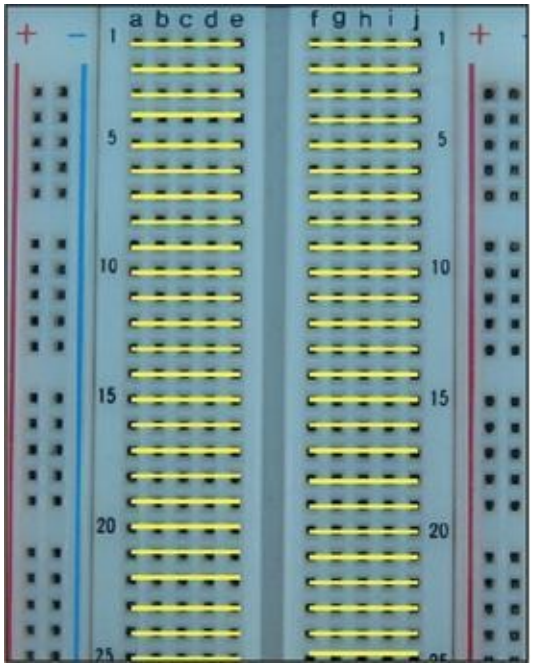


Figure 2.22 : Visualisation des liaisons de rangées.

Récapitulons

Dans ce chapitre, nous avons découvert :

- l’installation et la préparation de l’atelier Arduino ;
- la fenêtre d’édition du texte source ;
- la première compilation et le premier bogue ;
- quelques outils à se procurer.

Chapitre 3
Le tour de chauffe

AU MENU DE CE CHAPITRE :

- **Comment récupérer et implanter les exemples du livre**
- **Découvrir le Moniteur série**
- **Charger un programme déjà tout prêt**
- **Homéostasie ?**

Puisque ce livre est consacré à la programmation informatique, il est normal que tu y trouves de nombreux exemples de code source. Pour vérifier si ce que tu saisis est correct, ou lorsque tu es trop pressé, tu auras toujours la possibilité de charger la version complète de chaque projet.

Mais pour en profiter, il faut récupérer un fichier compressé, que l'on appelle une archive, au format ZIP sur le site de l'éditeur. Voyons cela sur-le-champ.

Accéder au site du livre

1. Ouvre ton navigateur Web et va à l'adresse suivante :

   `pourlesnuls.fr/livres/programmer-en-s-amusant-arduino-megapoche-pour-les-nuls-9782412023877`

   ou plus simple :

goo.gl/7xIEph

2. Tu arrives sur la page du livre. Tu y trouves une zone permettant de télécharger les fichiers.
3. Clique le(s) lien(s) de téléchargement.

pour les nuls
COLLECTION
LES LIVRES
AUTEURS
LES LIVRES
PROGRAMMER AVEC JAVASCRIPT EN S'AMUSANT MÉGAPOCHE POUR LES NULS
15 projets à réaliser dès 11 ans
Programmer avec JavaScript EN S'AMUSANT
Apprendre à coder en développant des projets amusants
Concevoir des jeux et créer une page Web
Animer un robot
PROGRAMMER AVEC JAVASCRIPT EN S'AMUSANT MÉGAPOCHE POUR LES NULS
de Eva HOLLAND et Chris MINNICK
NOUVEAU : La programmation enfin accessible aux enfants dès 8 ans grâce à 25 projets !
L'apprentissage de la programmation commence à être mise en place dans les établissements par l'Éducation nationale et cela depuis le plus jeune âge. De plus en plus d'ateliers de type "Kids Coding Clubs" voient le jour et comptent de plus en plus d'adeptes. Les livres de la collection Programmer en s'amusant sont destinés à la fois aux enfants désireux d'apprendre et les bases du codage informatique et également aux parents enseignants désireux d'accompagner leurs enfants ou leurs élèves dans la découverte d'un langage de programmation.
Ce livre est dédié à JavaScript, un langage de programmation ludique qui permettra aux enfants de 10 à 12 ans d'en apprendre les bases.
JavaScript est un langage de programmation gratuit qui fonctionne sur Mac et sur PC.
Ce livre contient plusieurs projets :
Concevoir des jeux et des applications Web
Développer soi-même une calculette
Créer un jeu inspiré de Madlibs
TÉLÉCHARGER LES DOCUMENTS
PROGRAMMER-EN-SAMU...
60 Ko
TÉLÉCHARGER
INFORMATIONS
Date de parution 10/12/2015
Nb. de pages 320
EAN13 papier 9782754081009
Format papier 210 x 148 cm
EAN13 numérique 9782754084093
Public : enfants de 10 à 12 ans, parents et enseignants.
17 €95
VERSION PAPIER
ACHETER
12 €99
VERSION NUMÉRIQUE
ACHETER

Figure 3.1 : La page Web d'un autre livre de la collection « Programmer en s'amusant » avec son lien de téléchargement.

Si l'adresse indiquée ci-dessus ne fonctionne pas, rends-toi sur la page pourlesnuls.fr puis clique l'icône de loupe et saisis les termes « programmer

arduino » dans la zone de recherche afin de trouver la fiche du livre (voir Figure 3.2).

4. Une fenêtre volante est apparue. Ouvre la liste pour sélectionner le ou les fichiers archives disponibles.

   Si tu n'as rien changé aux réglages de ton navigateur, le fichier va être stocké dans ton dossier des téléchargements. Si tu connais bien ton navigateur, tu as peut-être modifié une option pour que le programme te demande à quel endroit il doit déposer ce qu'il télécharge.

   Je suppose dans la suite que le fichier a été stocké dans `Téléchargements`.

Figure 3.2 : Recherche de la page du livre.

5. Une fois que le téléchargement est terminé, ouvre ton Explorateur de fichiers (appelé Finder sous Mac OS). Affiche les détails du dossier `Téléchargements`. Tu dois pouvoir repérer le fichier que tu viens de télécharger. Trie éventuellement l'affichage par dates décroissantes pour qu'il apparaisse en premier.

6. Clique du bouton droit dans le nom ou dans l'icône de l'archive pour choisir la commande **Couper**. Nous allons replacer l'archive dans le dossier destinataire.

   Toujours dans ton navigateur ou Explorateur de fichiers, ouvre le dossier `Documents`. Tu dois voir apparaître un sous-dossier portant le nom `Arduino`, qui a été créé lors de l'installation de l'atelier (Figure 3.3).

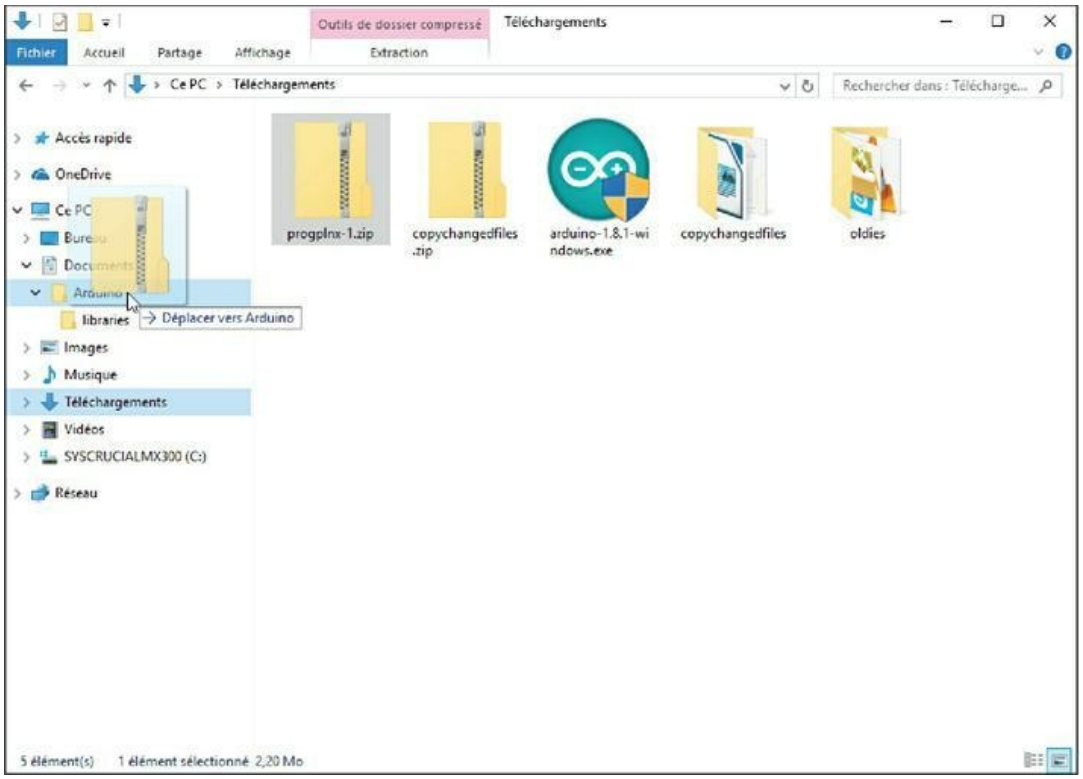


Figure 3.3 : Déplacement des exemples dans le dossier de travail Mes documents/Arduino.

7. Clique ce nom `Arduino` pour faire apparaître son contenu dans le volet des détails, puis amène le pointeur de souris dans une zone libre à droite et clique du bouton droit pour choisir la commande **Coller**. Tu peux également cliquer du bouton droit dans le nom du dossier `Arduino` et choisir **Coller**.

Le fichier archive est maintenant à l'endroit où il doit se trouver pour en déballer le contenu.

Décompression de l'archive des exemples

Clique du bouton droit le nom du fichier d'archive et choisis la commande **Extraire tout...**

Il ne reste plus qu'à patienter jusqu'à ce que tous les sous-dossiers soient créés et remplis avec les différents fichiers.

Dès que c'est fini, tu peux constater qu'il y a un sous-dossier par partie du livre. Parfois, tu te retrouves avec un sous-niveau en trop. Dans ce cas, redescends le « sous-sous-dossier » d'un niveau.

C'est quoi, ce dossier libraries ?

Tu as certainement remarqué qu'il y avait déjà un dossier dans `Arduino`. Ce dossier est vide pour l'instant. C'est ici que tu pourras déposer des fichiers archive de librairies, c'est-à-dire des collections de sous-programmes qui vont te faire gagner du temps. Nous verrons cela en temps utile.

Pour ne pas tomber chaos

Au fur et à mesure de ta progression dans ce livre, tu vas créer de plus en plus de programmes. Dans le monde Arduino, à chaque projet correspond un dossier. Tu vas donc rapidement avoir des dizaines de dossiers les uns à la suite des autres, ce qui est peu confortable.

Voilà pourquoi je te conseille de créer immédiatement dans ton dossier `Documents/Arduino` un premier sous-dossier auquel tu donneras le nom que tu veux, mais en évitant les espaces et les lettres accentuées. Par exemple : `ArduiTony` ou `GeekBoss` sont ok, mais pas `Mes projets d'été`.

Puisque tu envisages sérieusement de devenir un maître des machines, prends dès le départ l'excellente habitude de ne pas utiliser d'espace dans les noms de tes dossiers et de tes fichiers. C'est peut-être un peu gênant au début si tu n'en avais pas pris l'habitude, mais tu me remercieras lorsque tu auras besoin de travailler en mode texte dans un terminal, ce que je te souhaite un

jour.

Pour simuler l'espace, il suffit d'utiliser le caractère de soulignement ou underscore (le fameux « tiret_du_huit » sur le clavier) en remplacement de chaque espace. Le résultat visuel ressemble à ce que cela donnerait avec des espaces, comme dans cet exemple :

```
Mon très long nom de dossier
```

Voici le même sans espaces, ni accents :

```
mon_tres_long_nom_de_dossier
```

Nous en avons fini avec les préparatifs. En avant pour notre premier programme.

Découvrons le moniteur de l'atelier

Certains prétendent que la programmation informatique n'est pas un travail, mais un art. Il est vrai que la création d'un programme commence en général par une inspiration, et se poursuit avec beaucoup de transpiration mentale. Il faut avancer étape par étape, parfois un peu rebrousser chemin, et tout cela en observant comment se comporte le programme.

Mais pour savoir ce qui se passe dans la petite puce du mikon (le microcontrôleur, je le rappelle), il faut disposer d'un moyen pour que ce mikon renvoie des informations vers l'atelier. C'est à cela que sert le Moniteur série.

Le Moniteur série est une seconde fenêtre qui apparaît à côté de la fenêtre principale de l'atelier. Découvrons-la.

1. Pour démarrer l'atelier Arduino, clique son icône qui se trouve sur ton bureau, dans le dossier des applications

ou dans le menu général (selon le système dont tu disposes).

2. Lorsqu'il démarre, l'atelier recharge toujours tous les fichiers qui étaient ouverts au moment où tu l'as quitté la dernière fois. Si plusieurs fenêtres sont apparues, referme-les toutes sauf une. (Quand tu refermes la dernière fenêtre, cela fait quitter l'atelier.)

3. Pour ouvrir la fenêtre du Moniteur série, tu peux cliquer l'icône du bord droit en haut, celle qui représente une sorte de loupe avec un point au centre, ou bien utiliser la commande **Outils/Moniteur série** (Figure 3.4). L'endroit est tout à fait logique, puisque ce moniteur va devenir un de tes outils de travail pour la mise au point de tes projets.

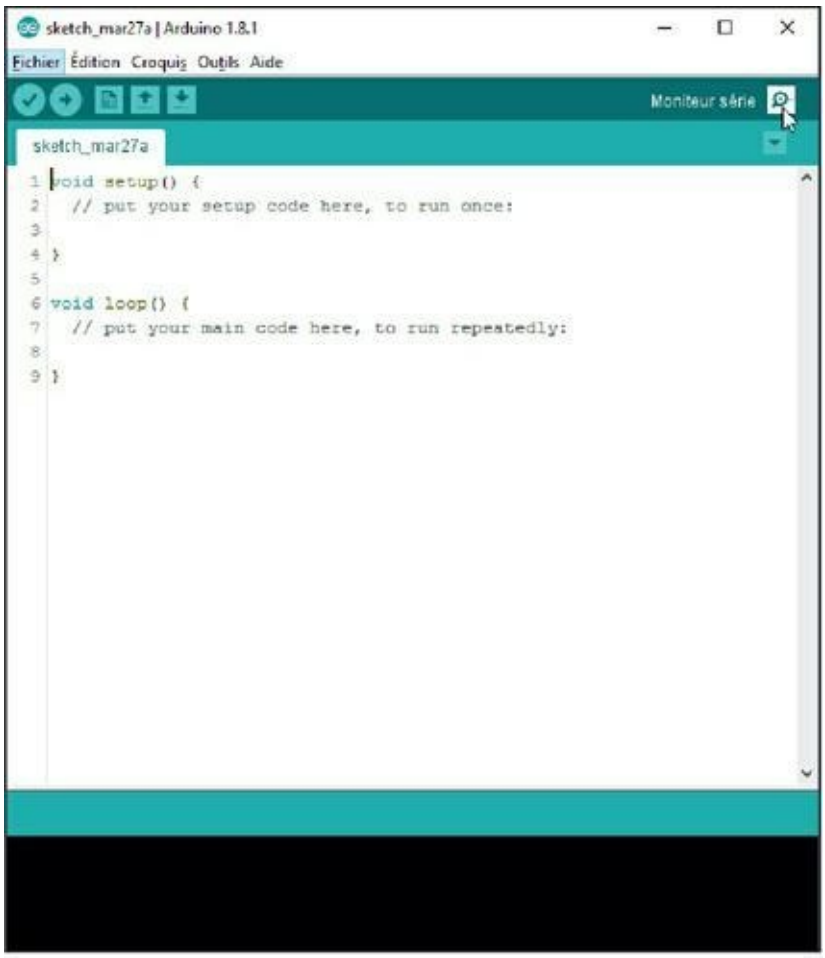


Figure 3.4 : Commande et icône pour ouvrir la fenêtre du Moniteur série.

4. Normalement, tu n'as pas encore connecté la carte Arduino *via* le câble USB. Autrement dit, si tout s'est bien passé, cela s'est mal passé. En effet, lorsque tu demandes l'ouverture de la fenêtre du Moniteur série et qu'il n'y a aucune carte Arduino détectée, un message d'erreur apparaît dans le panneau inférieur, comme quoi « *La carte sur COMx n'est pas disponible* » (Figure 3.5).

sketch_feb26a | Arduino 1.8.1
Fichier Édition Croquis Outils Aide
sketch_feb26a
void setup() {
// put your setup code here, to run once:
}
void loop() {
// put your main code here, to run repeatedly:
}
La carte sur COM3 n'est pas disponible
Recopier les messages d'erreur
La carte sur COM3 n'est pas disponible
La carte sur COM3 n'est pas disponible
Arduino/Genuino Uno sur COM3

Figure 3.5 : Le Moniteur série a besoin de détecter la carte Arduino !

Autrement dit, il ne reste plus qu'à brancher la carte avant d'essayer à nouveau d'ouvrir la fenêtre du moniteur.

5. Vérifie que ta carte Arduino n'est pas posée sur quelque chose de métallique, puis mets-la sous tension en branchant le câble USB.
6. Essaie à nouveau d'ouvrir la fenêtre du moniteur par le menu ou par l'icône. Cette fois-ci, elle devrait apparaître.
7. Repositionne les deux fenêtres de l'atelier pour qu'elles soient bien visibles toutes deux en permanence.

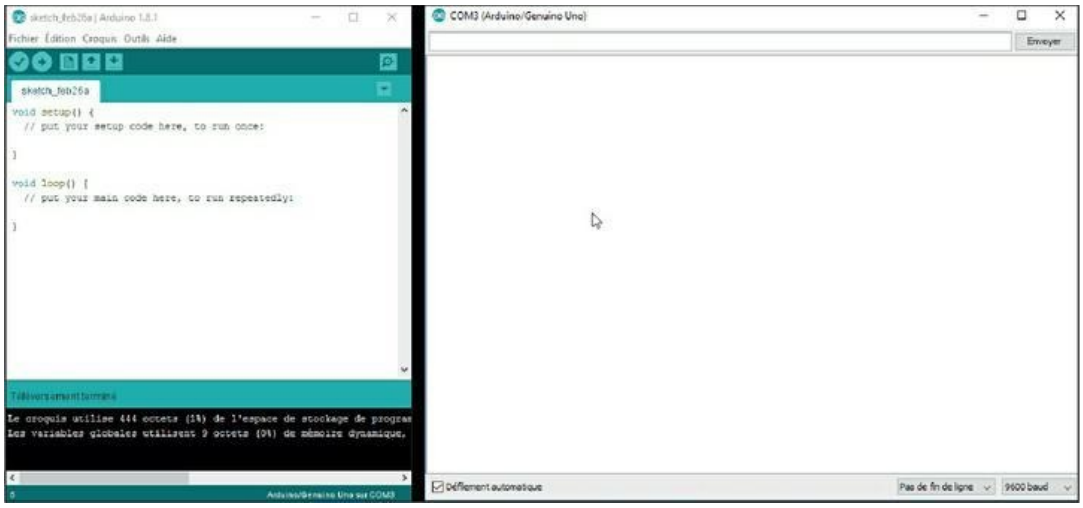


Figure 3.6 : Fenêtre du Moniteur série à droite de la fenêtre principale de l'atelier.

Tu es maintenant prêt à utiliser cet outil.

Chargement d'un programme

Un point très important concernant le Moniteur série : dans l'angle inférieur droit de la fenêtre du moniteur, tu vois deux listes de sélection. Celle de droite contient une valeur numérique exprimée dans l'unité Baud. Cette valeur est très importante (Figure 3.7).

Si, dans ton programme, tu insères une instruction pour configurer le moniteur avec une autre valeur que celle indiquée ici, ce qui va s'afficher n'aura aucun sens. Prends donc toujours l'habitude de vérifier ce qui sélectionné dans cette liste.

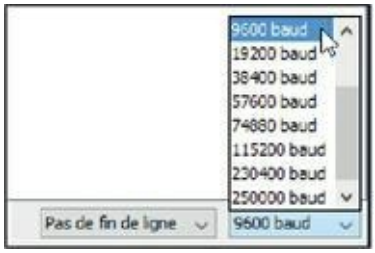


Figure 3.7 : Liste de sélection de la vitesse de communication.

Chargement du code source d'un programme

1. Dans la fenêtre principale de l'atelier, ouvre le menu **Fichier** et choisis **Ouvrir**.
2. Une boîte standard de sélection de fichiers apparaît. Utilise les possibilités de ton système pour naviguer parmi les dossiers afin d'atteindre celui dans lequel nous avons implanté tous les fichiers d'exemples, normalement un sous-dossier de `Documents/Arduino`.
3. Cherche le sous-dossier nommé **P1** (Figure 3.8). Dans ce sous-dossier, tu dois trouver un autre dossier portant le nom `P11_MaTempera`. Ouvre-le.

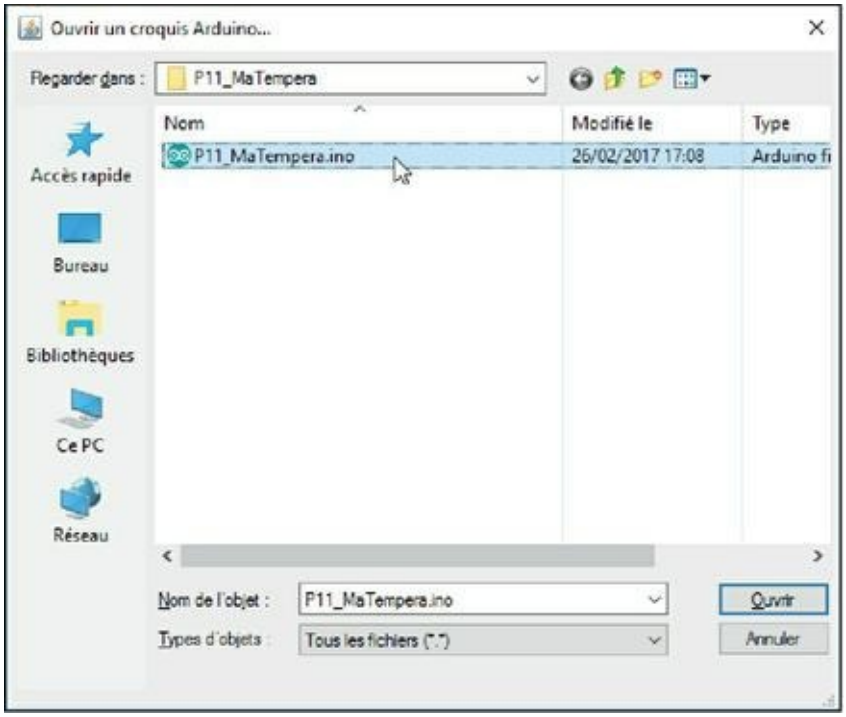


Figure 3.8 : Le dossier du premier exemple.

4. Tu vois alors un seul fichier portant le même nom que le dossier qui le contient. Il en va toujours ainsi avec l'atelier Arduino. Chaque projet est rangé de cette manière.

   Le fichier est le texte source du projet. Il se présente au format INO (extension de nom de fichier `.ino`). Il est possible que les extensions ne soient pas visibles dans le paramétrage de ta machine. Je te conseille fortement de les rendre visibles.

   Pour afficher les extensions, il faut accéder aux options de l'Explorateur de fichiers. Par exemple, sous Windows 10, il faut ouvrir le menu **Affichage** et cliquer **Options** sur le bord droit. Dans la boîte de dialogue des options des dossiers, choisis la deuxième page, **Affichage**. Dans la liste des options, descends un peu grâce à l'ascenseur à droite jusqu'à trouver l'option intitulée ainsi :

   Masquer les extensions des fichiers dont le type est connu.

   Active cette option en cliquant la case. Referme ensuite la boîte.

5 Il suffit de sélectionner l'unique fichier puis de cliquer le bouton **Ouvrir** en bas à droite. Tu peux aussi directement double-cliquer dans le nom de fichier ou son icône.

Passage en revue du code source

Tu dois maintenant avoir deux fenêtres d'édition ouvertes : celle qui est apparue quand tu as démarré le programme (l'embryon) et celle qui contient le nouveau fichier que nous venons de charger.

Si tu compares les contenus des fenêtres, tu dois pouvoir distinguer deux lignes identiques :

```
void setup()
{

void loop();
{
```

Ne prends pas peur en découvrant le contenu du programme d'exemple. Nous n'allons par le décortiquer, ni maintenant, ni plus tard. Contente-toi de repérer les deux lignes dont je viens de parler. Tu vas les trouver vers le début du contenu. Ce sont les noms des deux fonctions qui doivent toujours être présentes dans un projet Arduino, dans tous les cas. Nous verrons dans le chapitre suivant à quoi servent ces lignes.

Je t'accorde volontiers que nous démarrons sur les chapeaux de roue. En effet, ce premier programme contient des techniques que la plupart des passionnés de l'Arduino ne découvrent que bien plus tard, et certains jamais.

Mais ici, il s'agit d'un livre consacré à la programmation et non simplement au branchement d'un capteur ou d'un moteur suivi de la récupération d'un programme déjà écrit par quelqu'un d'autre. Donc, on bombe le torse, et en avant !

Vérification et compilation

Pour faire bonne mesure, nous allons lancer une vérification, qui doit normalement se dérouler sans erreur. Nous lançons ensuite le programme en le transférant vers le mikon de la carte Arduino.

1. Tu vois la fenêtre d'édition et la fenêtre du moniteur ? Bien. Dans l'atelier, utilise la commande **Vérifier**, soit en cliquant l'icône la plus à gauche en haut, soit en ouvrant le menu **Croquis** puis en choisissant **Vérifier/Compiler** (Figure 3.9).

Figure 3.9 : Commandes du menu Croquis pour vérifier et téléverser le projet.

2. Observe la partie inférieure. Tu vois la compilation en cours. Une fois celle-ci réussie, deux lignes apparaissent (Figure 3.10).

   La première ligne indique l'occupation mémoire du code machine qui correspond à ce programme source. Normalement, la valeur est égale à 3536. J'indiquais dans le premier chapitre que le mikon de l'Arduino Uno possédait un espace mémoire Flash pour les programmes. Cette mémoire Flash offre 32 ko, soit 32 768 octets. Nous en avons consommé presque 10 %.

P11_MaTempera | Arduino 1.8.1
Fichier Édition Croquis Outils Aide
P11_MaTempera
// P11_MaTempera
// Lecture de la température interne du mikon
// CEN 201703 V1.0
//////////////////
void setup()
{
Serial.begin(9600);
Serial.println(F("Temperature interne du micro-controleur !"));
}
void loop()
{
unsigned int wLecture;
double temp;
// Configure ref interne et le mux.
ADMUX = (_BV(REFS1) | _BV(REFS0) | _BV(MUX3));
ADCSRA |= _BV(ADEN); // Active le convertisseur A/N
delay(20); // Petite pause pour stabiliser.
ADCSRA |= _BV(ADSC); // Lance la conversion...
// ...et attend qu'elle soit finie
while (bit_is_set(ADCSRA,ADSC)) ;
// Recupere les deux octets.
wLecture = ADCW; // Registre sur 16 bits, un mot (WORD)
// Convertit de kelvin en Celsius (approximatif).
temp = (wLecture - 330) / 1.22;
Compilation terminée
Le croquis utilise 3536 octets (10%) de l'espace de stockage de programmes. Le maximum est de 32256
Les variables globales utilisent 214 octets (10%) de mémoire dynamique, ce qui laisse 1834 octets po

Figure 3.10 : Messages de compilation réussie.

Utilise si nécessaire le translateur (ascenseur horizontal) sous le panneau inférieur pour voir la fin du message.

La deuxième ligne parle de variables globales. Tiens, nous n'avons pas encore vu ce qu'était une variable globale (ni pas globale d'ailleurs). Ce sont les emplacements mémoire dont le programme a besoin pour stocker les valeurs qu'il manipule. Ici, nous en consommons 214 octets.

3. Si la compilation réussit, l'image exécutable du programme est copiée vers la mémoire du mikon sur la

carte, puis le programme démarre immédiatement.

Dans la fenêtre du Moniteur série, la température indiquée doit se situer vers les 21 °C (Figure 3.11). Si tu viens de démarrer la carte Arduino, c'est à peu près la température de la pièce. Note cependant que le mikon n'est pas un thermomètre. Nous interrogeons un capteur interne qui permet d'éviter au mikon d'être détruit par une trop forte chaleur. Il ne peut pas servir de thermomètre.

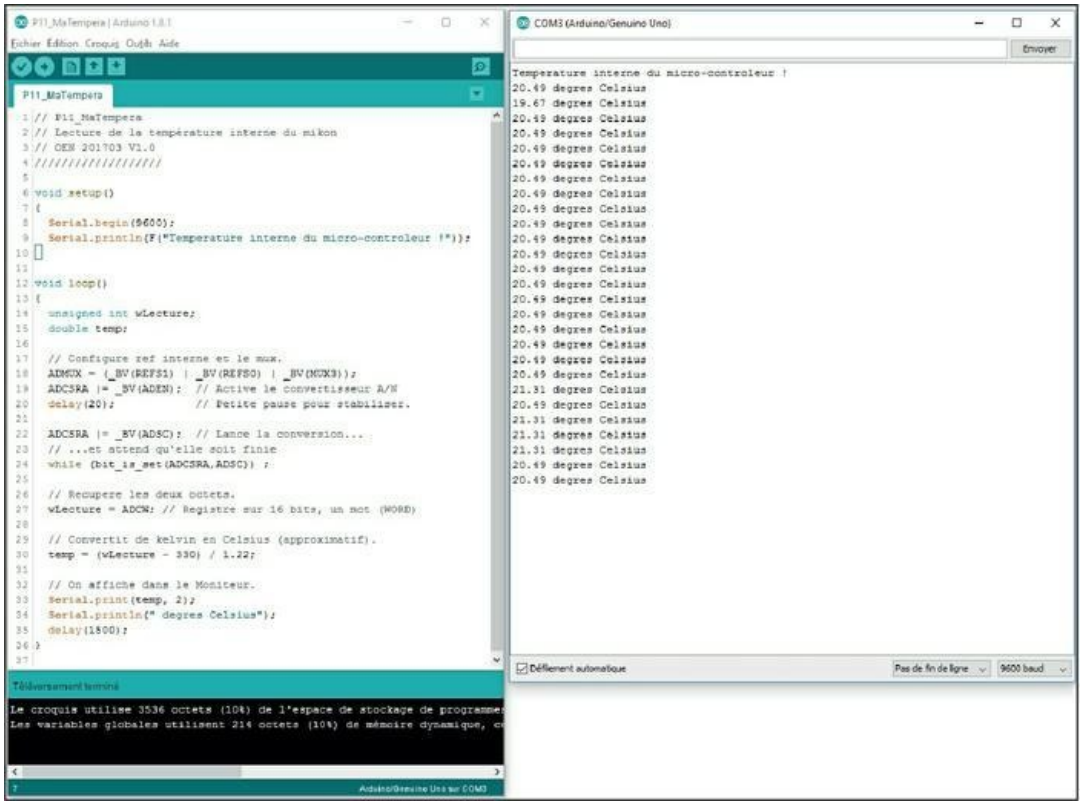

P11_MaTempera | Arduino 1.8.1
Fichier Édition Croquis Outils Aide
P11_MaTempera
// P11_MaTempera
// Lecture de la température interne du mikon
// OEN 201703 V1.0
void setup()
{
Serial.begin(9600);
Serial.println(F("Temperature interne du micro-controleur !"));
}
void loop()
{
unsigned int wLecture;
double temp;
// Configure ref interne et le mux.
ADMUX = (_BV(REFS1) | _BV(REFS0) | _BV(MUX3));
ADCSRA |= _BV(ADEN); // Active le convertisseur A/N
delay(20); // Petite pause pour stabiliser.
ADCSRA |= _BV(ADSC); // Lance la conversion...
// ...et attend qu'elle soit finie
while (bit_is_set(ADCSRA,ADSC)) ;
// Recupere les deux octets.
wLecture = ADCW; // Registre sur 16 bits, un mot (WORD)
// Convertit de kelvin en Celsius (approximatif).
temp = (wLecture - 330) / 1.22;
// On affiche dans le Moniteur.
Serial.print(temp, 2);
Serial.println(" degres Celsius");
delay(1800);
}
Téléversement terminé
Le croquis utilise 3536 octets (10%) de l'espace de stockage de programme
Les variables globales utilisent 214 octets (10%) de mémoire dynamique,
COM3 (Arduino/Genuino Uno)
Envoyer
Temperature interne du micro-controleur !
20.49 degres Celsius
19.67 degres Celsius
20.49 degres Celsius
Défilement automatique
Pas de fin de ligne
9600 baud

Figure 3.11 : Affichage de la température du mikon dans le Moniteur série.

4. Note que le Moniteur série continue à afficher toujours la même température avec parfois quelques petits

sauts.

Voyons si nous pouvons influencer physiquement la température du mikon.

5. Sans hésiter, pose un doigt (non mouillé !) ou ton pouce sur le boîtier noir du micron et appuie légèrement pour transmettre un peu de chaleur de ton corps. Observe bien le Moniteur série. Tu devrais voir monter la température. Change éventuellement de doigt pour transmettre encore plus de chaleur.

   En appuyant suffisamment, mais sans forcer, tu dois pouvoir faire monter la température d'un ou de deux degrés environ (Figure 3.12). Observe l'affichage du Moniteur série.

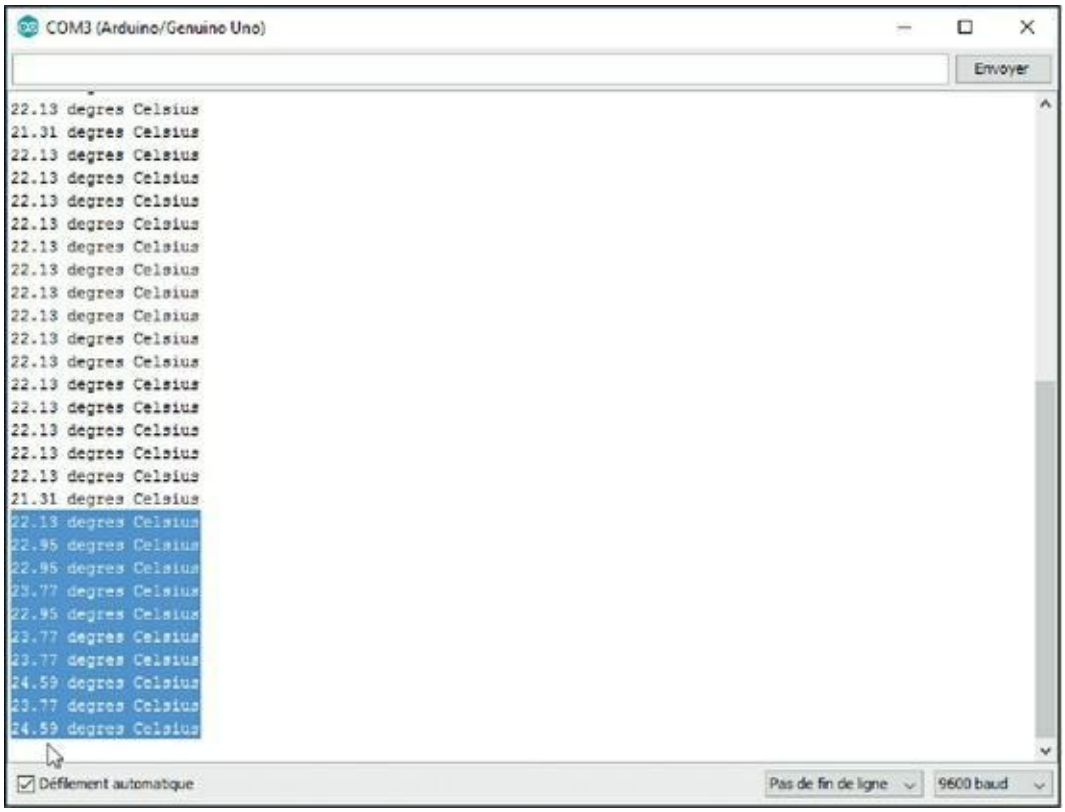


Figure 3.12 : Faisons chauffer la bête !

6. Retiens si nécessaire la carte par son câble USB et souffle sur le mikon pour le refroidir (sans postillonner dessus, si possible). Normalement, tu dois pouvoir faire redescendre la température du mikon. Il réagit donc bien à ta présence.

Avec ce petit exemple pratique, tu es entré en contact physique avec ton mikon. Partons maintenant à la découverte du langage avec lequel tu vas créer la pensée d'Arduino.

Récapitulons

Dans ce chapitre, nous avons appris à :

- télécharger les fichiers d'exemples ;
- ouvrir la fenêtre du Moniteur série ;
- charger un programme ;
- vérifier et téléverser un programme ;
- lire et modifier la température interne du mikon.

Semaine 2 : Parler et écouter le monde

Au menu de cette semaine :

Chapitre 4 : À l'écoute du monde en noir et blanc ?

Chapitre 5 : À l’écoute du monde réel

Chapitre 6 : Parler, c’est agir

Chapitre 7 : Donner une impulsion

Chapitre 4
À l’écoute du monde en noir et blanc

AU MENU DE CE CHAPITRE :

- **L’art d’écrire un texte source**
- **Détecter l’état d’une entrée numérique**
- **Afficher une valeur dans le Moniteur série**
- **Éviter les flottements avec PULLUP**

Le Web a démarré de façon expérimentale en 1994 pour ce qui est des pays francophones. Comme sur toute terre inconnue, il a fallu apprendre sur le tas. La surprise a été au rendez-vous.

Nous atteignons ici un grand moment dans le cours de ce livre : nous allons apprendre en détail à écrire un texte pour animer un microcontrôleur.

Partir d’une feuille blanche (ou presque)

Pour l’instant, nous n’aurons pas besoin de la carte Arduino. Débranche-la si nécessaire.

1. Démarre l'atelier comme tu as appris à le faire au cours du chapitre précédent, en utilisant l'icône ou le nom de l'application dans le menu.

   Lorsque la fenêtre d'édition de l'atelier apparaît, il est possible qu'elle présente le dernier projet sur lequel tu as travaillé. Ce n'est qu'au premier démarrage que tu vois apparaître directement le fichier quasi vide qui sert de point de départ.

2. Ouvre le menu **Fichier** pour choisir la commande **Nouveau**.

   Tu vois apparaître une seconde fenêtre d'édition. C'est celle qui nous intéresse.

3. Referme la première fenêtre qui est apparue au démarrage (sauf si c'est ton premier démarrage). Utilise la commande **Fichier/Fermer** de la fenêtre à fermer ou la case de fermeture habituelle (en haut à droite ou à gauche).

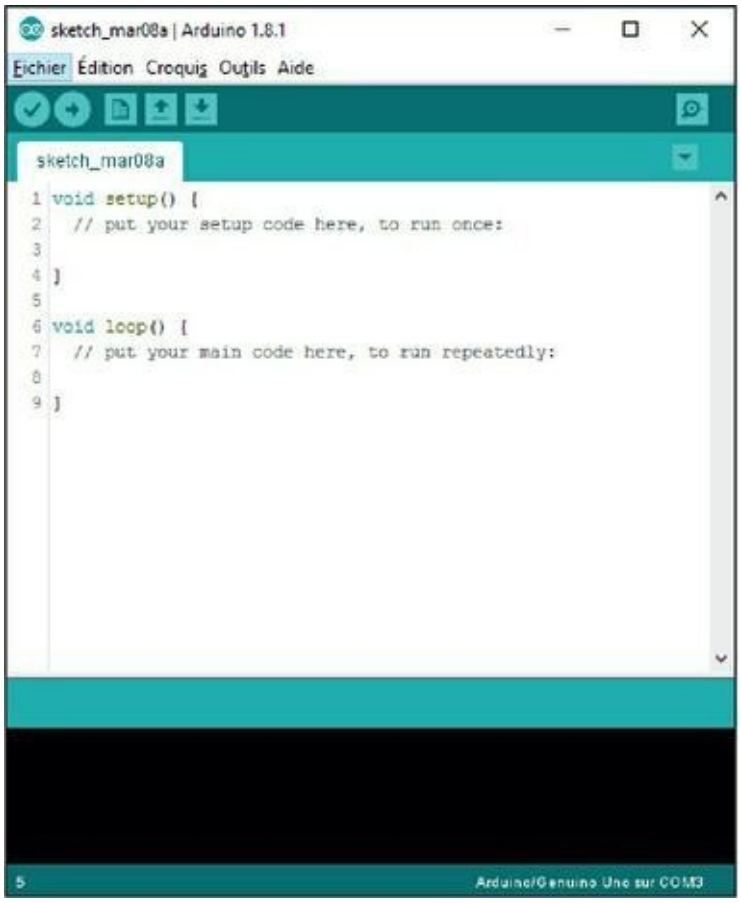


Figure 4.1 : État initial de l'éditeur au démarrage d'un nouveau projet.

Si tu compares le résultat à ce que l'on a l'habitude de voir quand on commence avec un nouveau fichier, par exemple dans un traitement de texte, tu peux t'étonner de voir que la fenêtre n'est pas vide. Pourquoi ?

La réponse demande de passer aux présentations. Le langage dans lequel tu écris tes projets Arduino est une variante d'un des langages les plus répandus dans le monde entier, le langage C.

Le langage C ?

Le langage C est une sorte de patriarche dans les info-langages. Il a eu une nombreuse descendance, et notamment les langages

C++, Java et C#. Créé dans les années 1970, le langage C a trouvé un bel équilibre entre deux besoins opposés : celui de travailler vite en ne se perdant pas dans les détails, et celui de contrôler précisément la partie matérielle en entrant justement dans tous les détails.

Ce que tu vois dans la fenêtre de l'éditeur est un embryon de programme, en anglais un stub. Il contient deux définitions de fonctions, ces deux fonctions devant être présentes dans tout projet Arduino. Voyons cela dans un listing.

Listing 4.1 : L'embryon de programme Arduino.

```
void setup() {
  // Ici viennent les instructions de
préparation
}

void loop() {
  // Ici celles qu'il faut répéter
}
```

Que voit-on au juste ?

- Il y a deux lignes qui commencent par un mot anglais, void, qui signifie « vide », et se poursuivent avec un nom suivi d'une paire de parenthèses et d'une accolade ouvrante.
- Il y a deux lignes qui commencent par un double signe /. Ces lignes sont totalement ignorées par la

> vérification et la compilation. Elles ne servent qu'à l'auteur du programme et à ses collègues.
>
> » Et, enfin, deux lignes qui ne contiennent qu'une accolade fermante.

Tu devines déjà que les signes de ponctuation vont avoir une énorme importance. Par exemple, les accolades doivent toujours aller par deux. S'il y en a une ouvrante, il faut quelque part que l'on trouve la fermante qui lui correspond.

Savoir faire bloc

En langage C, les accolades servent à marquer le début et la fin d'un bloc autonome. Toutes les lignes d'instructions qui seront placées entre les deux accolades seront exécutées de la première à la dernière, mais l'exécution ne se poursuivra pas toujours après la fermante.

La lecture normale d'un programme ressemble à celle d'un texte humain : on commence par le début et on descend de ligne en ligne jusqu'à la fin. Avec un jeu d'accolades, cette lecture est limitée au début et à la fin du bloc. Pour l'instant, les deux blocs de l'embryon sont vides. Les lignes de commentaires (en anglais dans l'original) n'existent pas pour le mikon. Vérifions cela.

Mise en pratique : première vérification

La commande **Vérifier** est utilisable même lorsque la carte Arduino n'est pas connectée (mais pas la suivante, **Téléverser**). Nous allons profiter de cette possibilité.

1. Utilise le bouton **Vérifier** et observe le panneau inférieur de l'éditeur.

Au bout de quelques instants, tu dois voir apparaître les deux messages de fin de vérification qui rappellent l'espace occupé par les instructions du programme et celui occupé par les variables globales.

En fait, dans cet état, le projet ne fait absolument rien, mais il est valable et accepté par le vérificateur. Nous pourrions donc même le compiler et le téléverser dans la carte Arduino, qui ne s'en plaindrait pas. Simplement, le programme tournerait en boucle sans rien faire.

Le fait que les deux fonctions soient totalement vides n'a aucun rapport avec la présence du mot qui signifie « vide » en début de ligne (void).

À quoi sert une fonction ?

Une fonction, c'est quelque chose qui fonctionne, qui réalise un certain travail. En informatique, ce travail est appliqué à des données. Le nom « fonction » provient au départ du monde des mathématiques. En mathématiques, une fonction peut s'écrire de la façon suivante :

```
F(x) = x + 2
```

Dans cet exemple, la fonction qui porte le nom F effectue un traitement qui consiste à multiplier la valeur par deux. Si on utilise cette fonction avec la valeur 4 pour la variable x, la fonction vaudra 6.

D'ailleurs, le couple de parenthèses qui suit immédiatement les deux mots `setup` et `loop` témoigne de cet héritage. Ici, ce couple est vide, mais dans tes projets, tu indiqueras entre les deux parenthèses le nom d'une ou de plusieurs variables qui vont contenir des valeurs ; la fonction va travailler sur ces valeurs.

En informatique, la notion de fonction est plus vaste qu'en mathématiques. Tu pourras créer des fonctions qui n'auront pas besoin de renvoyer le résultat de leur travail. Pour les deux fonctions obligatoires, c'est le cas, et cela explique le mot void.

Par exemple, imaginons une fonction que nous décidons de nommer `controlerMoteur()` (tu notes que je n'utilise pas de lettres accentuées dans les noms des fonctions, même en français, car c'est interdit).

Cette fonction pourra, par exemple, démarrer et arrêter un moteur électrique. Pour te servir de la fonction, tu vas indiquer son nom ailleurs dans le programme, c'est-à-dire appeler la fonction.

Appeler une fonction, c'est demander son exécution immédiate.

Tu comprends donc que tout programme doit contenir au moins une fonction. Dans le langage C normalisé, cette fonction obligatoire porte le nom anglais qui signifie « principal », c'est-à-dire `main()`. Dans la variante Arduino, il a été choisi de prévoir dès le départ non pas une, mais deux fonctions. En quoi se distinguent-elles ?

Les deux fonctions setup() et loop()

Tout projet demande normalement quelques préparatifs, comme le choix du mode d'utilisation d'une broche, en entrée ou en sortie, le réglage d'une vitesse de dialogue avec un ordinateur, la préparation d'une variable, etc. Cette étape de configuration ne doit par définition être réalisée qu'une seule fois au démarrage du programme. Et le mot anglais pour configurer est setup. Voilà pour la première des deux fonctions.

Une fois les préparatifs terminés, il reste plus qu'à se mettre au travail ! C'est à ce niveau qu'il y a une énorme différence entre des

projets pour un microcontrôleur, tel que celui de la carte Arduino, et les applications pour les plus grands ordinateurs.

Le mikon de ta carte ne peut par définition exécuter qu'un seul programme à la fois et il l'exécute sans jamais s'arrêter. Il n'y a pas de commande Quitter dans un projet Arduino. C'est ce qui explique le nom choisi pour la deuxième fonction obligatoire : en anglais, loop signifie « boucle » . Toutes les instructions que tu vas ajouter entre les deux accolades du bloc de cette fonction seront exécutées de la première à la dernière, puis à nouveau de la première à la dernière, et ainsi de suite jusqu'à ce que tu coupes l'alimentation électrique de la carte ou que tu utilises le bouton de réinitialisation Reset.

À l'intérieur de la fonction `loop()`, tu peux non seulement insérer des instructions, par exemple pour additionner deux nombres, mais tu vas surtout pouvoir y placer des appels vers d'autres fonctions. C'est à partir de cette possibilité que se déploie toute la puissance de la programmation informatique.

Je t'accorde que les précédents paragraphes ont été assez touffus, et je te propose un peu de mise en pratique en guise de récréation.

Mise en pratique : allègement du code source

1. Pour l'instant, le projet porte un nom automatique, peu suggestif : le mot sketch, un signe _ et le début du nom du jour et son numéro.

   Choisis un joli nom pour ce projet, par exemple `CH04_Digit1`.

2. Clique dans la ligne de commentaires qui constitue le corps du bloc de la première fonction et supprime toute cette ligne.

Une première solution consiste à triple-cliquer dans la ligne puis utiliser la touche **Suppr**. Tu peux également te placer contre la marge gauche et sélectionner par glissement.

3. Fais la même chose pour supprimer la ligne de commentaires dans la fonction principale `loop()`.

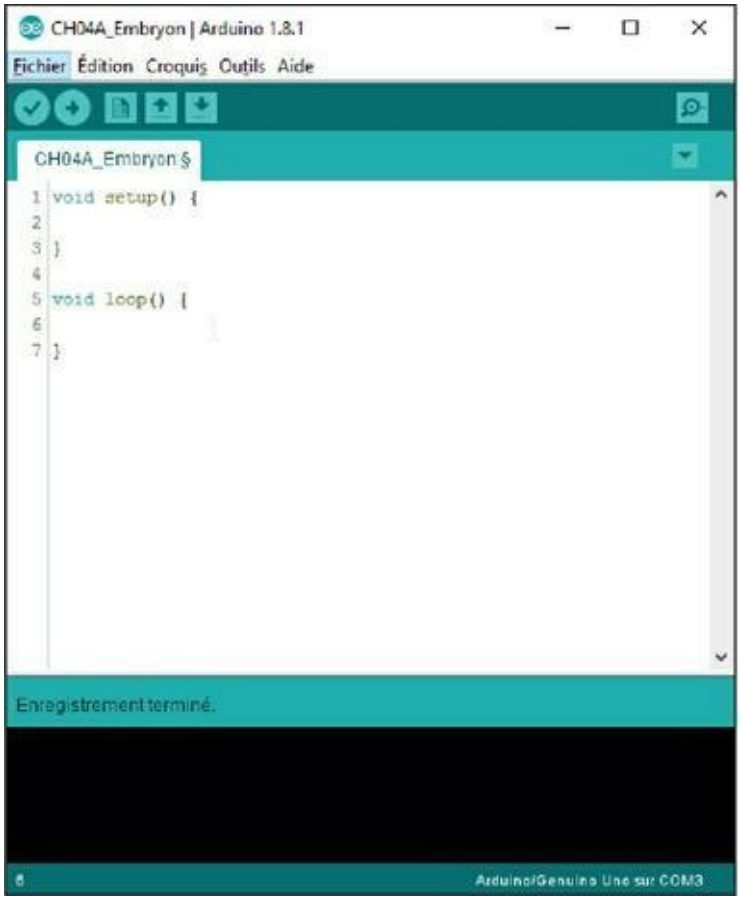


Figure 4.2 : L'embryon de programme allégé.

4. Lance une vérification comme tu sais le faire maintenant. Il ne devrait y avoir aucune différence dans le résultat.

5. Essayons maintenant de jouer avec un délimiteur de bloc. Place-toi juste à droite de la première accolade et supprime-la avec la touche **Retour arrière** (ou utilise une autre technique pour supprimer ce caractère).

   Dès que tu modifies l'embryon, tu es obligé de stocker le texte source dans un fichier sur disque.

6. Lance une nouvelle vérification qui, si tout va bien, devrait mal se passer.

   Si tu observes le panneau inférieur, tu dois voir des messages en orange. Le compilateur se plaint parce qu'il est perdu. La simple absence d'une accolade dérègle toute la structure du programme.

7. Replace l'accolade là où elle était, ce qui te permet de prendre tes repères sur le clavier.

Mais où sont les accolades ? C'est simple :

> » Sous Windows, il faut combiner la touche **Alt Gr** avec les touches du dessus **'4** ou **=+**, comme sous Linux.
>
> » Sous Mac OS, il faut utiliser les trois touches **Alt**, **Maj** et la bonne parenthèse.

Nous en savons assez pour nous lancer dans notre premier projet pratique.

Projet 1 : détection sur une entrée numérique

Comme un animal, un processeur s'inscrit dans son environnement. Il a des organes des sens pour savoir ce qui se passe autour de lui ; il a des nerfs moteurs pour agir sur son environnement. Intéressons-nous d'abord à ses organes des sens, ses nerfs sensitifs. En langage informatique, cela correspond à ses entrées (inputs).

Nous allons commencer en douceur en observant ce qui se passe sur une broche d'entrée numérique du mikon. Tu as appris dans les chapitres antérieurs que le processeur de la carte Arduino Uno possède 14 broches numériques qui peuvent servir en entrée comme en sortie. Nous allons brancher sur une de ces broches un simple fil muni de broches mâles, ce que l'on appelle un strap. Il servira d'antenne pour que la valeur lue varie de façon imprévisible.

On ne peut pas faire plus simple que ce premier projet, mais cela ne nous empêche pas de prendre dès le départ de bonnes habitudes. Il faut commencer par une analyse théorique, pour définir les contours de ce qu'on appelle un algorithme. Tu vas voir que c'est très facile.

Analyse théorique de la solution

Le problème à résoudre est très simple : nous devons pouvoir détecter le passage de l'entrée numérique de l'état 0 ou à l'état 1. Nous allons donc demander au mikon de lire de façon répétée la valeur qu'il trouve sur la broche d'entrée numérique que nous allons lui indiquer.

Mais comme les broches numériques peuvent servir en entrée comme en sortie, il faut d'abord lui dire que nous avons décidé d'utiliser la broche choisie comme entrée, et non comme sortie.

La Figure 4.3 montre la logique de notre projet dans sa première version.

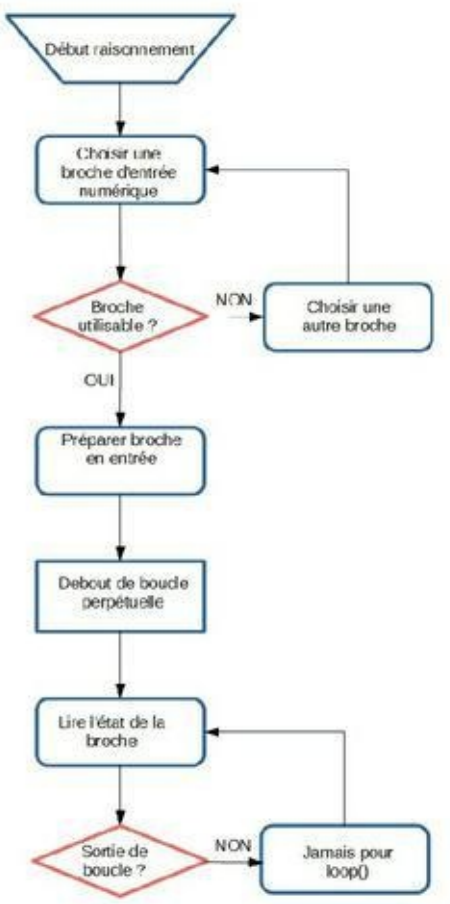


Figure 4.3 : Solution du problème de lecture d'une entrée numérique.

L'inné et l'acquis

Le mikon possède dès le départ un certain nombre de fonctions internes, dont une fonction pour lire l'état d'une broche numérique vers un registre, et une autre pour copier la valeur depuis ce registre vers un emplacement dans la mémoire. Ces fonctions sont dites natives : ce sont des instructions machine, gravées à tout jamais dans la matière de la puce. Elles sont très élémentaires, et cela nous prendrait beaucoup de temps d'écrire tous les détails des opérations requises uniquement avec elles.

En effet, avant de procéder à la lecture, il faut préparer le port puis le mikon lui-même et réaliser différentes opérations dans les mémoires internes du mikon pour aboutir au résultat.

Le jeu d'instructions d'un microcontrôleur correspond à la partie innée, la même qui permet à un bébé animal de se lever dès la naissance, sans avoir encore acquis aucun savoir du monde extérieur.

Chacun des projets que tu vas créer correspondra à l'acquis du mikon, ce que tu vas lui apprendre à faire. Entre ces deux extrêmes (l'inné et l'acquis), il y a tout un monde intermédiaire qui est celui des fonctions prédéfinies. Elles ont été conçues par d'autres personnes. Tu peux en profiter directement. Pas besoin de les écrire ; cela te fera gagner énormément de temps.

Nous allons utiliser dans tout ce livre de nombreuses fonctions dites prédéfinies. Ces fonctions sont regroupées par thèmes dans des conteneurs que l'on appelle des librairies (anciennement bibliothèques). Il y a ainsi une librairie pour émettre des sons, une autre pour piloter un circuit GPS, etc.

Pour être encore plus précis, il y a un noyau d'environ 60 fonctions qui sont installées dès le départ en même temps que l'atelier Arduino. Les autres devront être téléchargées et installées en fonction de tes besoins.

Passons à la pratique en utilisant deux premières fonctions prédéfinies : la première va préparer le mikon à utiliser la broche choisie comme entrée numérique, et la seconde va lire la valeur qui se trouve à ce moment sur la broche.

Les noms des fonctions prédéfinies sont quasiment toujours en anglais. Celles dont nous allons avoir besoin sont les suivantes :

```
pinMode()
digitalRead()
```

La fonction pinMode()

La fonction standard qui permet de choisir le mode d'utilisation d'une broche (pin en anglais), `pinMode()`, a besoin, pour travailler correctement, qu'on lui transmette deux valeurs au démarrage, c'est-à-dire deux paramètres d'entrée :

- le numéro de la broche concernée, ici la broche numéro 7 ;
- un mot réservé indiquant le mode d'utilisation, soit INPUT, soit OUTPUT (avec des variantes que nous verrons plus tard). Ici, ce sera INPUT.

Puisque le réglage du mode n'a besoin d'être réalisé qu'une seule fois au début, nous allons ajouter cette première instruction dans le bloc (le corps) de la première des deux fonctions obligatoires, c'est-à-dire `setup()`.

1. Dans l'éditeur Arduino, clique entre les deux accolades, et insère s'il le faut une ligne vide avec la touche **Entrée**. Tu es dans le corps de la première fonction.
2. Saisis la suite de caractères suivante, sans en omettre un seul. Tu remarques les deux espaces au début, sans doute insérées d'office par l'éditeur :

   ```
   pinMode(7, INPUT) ;
   ```

3. Procède de la même façon dans le corps vide de l'autre fonction obligatoire, `loop()`, en saisissant exactement l'instruction suivante. Entre parenthèses, c'est le numéro de la broche d'entrée que la fonction doit lire :

   ```
   digitalRead(7) ;
   ```

4. Prends maintenant quelques instants pour relire le contenu de ton texte source. Il doit correspondre exactement au listing suivant.

   Tu peux stocker la nouvelle version sous un nouveau nom. Elle correspond à l'exemple `CH04_Digit2`.

Listing 4.2 : Deuxième version du projet Digit (CH04_Digit2)

```
void setup() {
  pinMode(7, INPUT);
}

void loop() {
  digitalRead(7);
}
```

Préparation du montage physique

Notre programme est prêt à être testé, mais la carte Arduino pas encore.

1. Observe bien ta carte Arduino. Repère le connecteur noir du haut, celui marqué DIGITAL (PWM). Il est en deux parties : numéros 0 à 7 puis numéros 8 à 13 (ainsi que quelques broches sans intérêt pour l'instant, sauf GND).

2. Sans encore brancher la carte à l'ordinateur, insère un strap de quelques centimètres de long dans la broche

numérique numéro **7**. Aide-toi de la photo en Figure 4.4.

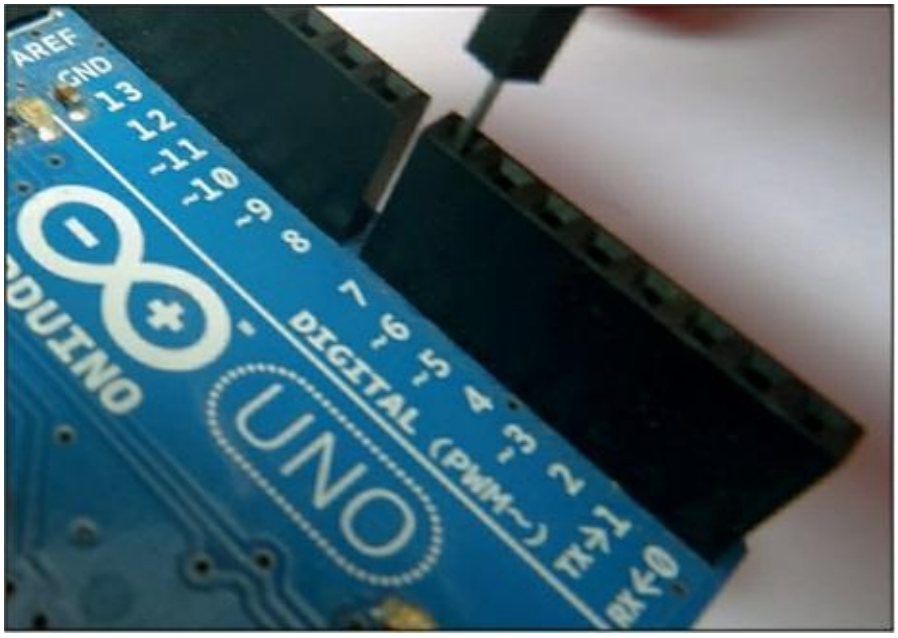


Figure 4.4 : Branchement d'un strap sur la broche numérique.

3. Vérifie que la broche est bien enfoncée et que le bout libre ne touche rien de métallique, ni sur la carte, ni là où tu l'as posée (Figure 4.5). Nous aurons besoin de repositionner ce bout libre pour faire varier la valeur lue.

Figure 4.5 : Le bout libre du strap doit pendre dans le vide.

4. Branche le câble USB sur l'ordinateur.

Test du projet

Nous pouvons maintenant téléverser le projet dans la mémoire du programme du mikon sur la carte.

1. Commence par un dernier contrôle au moyen de la commande **Vérifier**. Le programme doit se compiler correctement.

S'il y a un problème, relis le texte source en le comparant au contenu du listing précédent.

2. Utilise maintenant la commande **Téléverser** pour transférer le code binaire résultant de ce texte source dans le mikon et en lancer l'exécution.

   Que se passe-t-il ? Il ne se passe rien du tout. Pourtant, je t'assure que le programme fonctionne et que le processeur n'arrête pas de lire la valeur qu'il trouve sur la broche à laquelle nous avons relié le strap. Que faut-il faire ?

   Il nous faut récupérer la valeur lue en donnant un nom à une case dans la mémoire, case dans laquelle nous allons entreposer cette valeur. En fait, nous allons créer notre première variable.

   Mais auparavant, un petit retour sur les fonctions.

À propos des noms des fonctions

Nous venons de saisir deux noms de fonctions, `pinMode()` et `digitalRead()`. Quelques remarques :

- Tout d'abord, il n'y a jamais d'espace dans un nom (on parle aussi souvent d'*identifiant* au lieu de nom).
- Deuxièmement, tu constates que l'on utilise une manière d'écrire assez originale dans les noms, c'est l'écriture en dos de chameau (ou droMaDaire). Le nom commence toujours par une minuscule, mais les mots accolés commencent chacun par une capitale pour en

> faciliter la lecture : `pinMode()` est la réunion des deux mots anglais *input* (entrée) et *mode*. De même, `digitalRead()` est l'union du mot *digital* (en vrai français, on devrait d'ailleurs dire « numérique ») et *read* qui est le verbe « lire ».

En écrivant un tel mot suivi entre parenthèses des noms des variables d'entrée, tu demandes d'exécuter les instructions du corps de la fonction. Tu ne vois pas ces instructions. La fonction peut contenir des dizaines ou des centaines d'instructions. Cela te fait gagner un temps énorme, en profitant du travail des autres programmeurs.

Notre première variable

Une variable est un nom, mais à la différence de celui d'une fonction, qui provoque une action, le nom d'une variable sert à ranger et à retrouver une valeur, une donnée, dans la mémoire des données du mikon. Nous avons vu dans un chapitre antérieur que les cartes telles que Arduino possèdent une très petite quantité de mémoire pour les données. Par exemple, il n'y a que 2048 octets disponibles dans le mikon ATmega328 qui équipe la Arduino Uno. Autrement dit, il ne faut jamais réserver plus d'espace mémoire que nécessaire pour stocker tes variables.

C'est pour cela qu'il existe des mots réservés dans le langage Arduino, le langage C, pour décider de la quantité d'espace mémoire qu'il faut réserver au stockage de chaque donnée du programme. Dans le cas de l'entrée numérique, il n'y a que deux possibilités pour la valeur : soit 0, soit 1 (c'est-à-dire soit 0 volt, soit 5 volts). Le langage C offre une dizaine de tailles différentes, que l'on appelle des types de données. Pour notre projet, le plus petit te suffira ; il porte le nom `boolean`.

Comme dans les hôtels, avant de pouvoir y dormir, il vaut mieux réserver. En langage C, avant de pouvoir stocker une valeur, il faut

réserver l'espace, ce qui correspond à la phase de déclaration de la variable.

Voici comment va s'écrire la déclaration de notre première variable :

```
boolean bValeur;
```

Mais où placer cette nouvelle ligne ? Dans la première ou dans la deuxième fonction ? Réponse : ni dans l'une, ni dans l'autre. Nous allons déclarer la variable au début de notre texte source, avant le mot `void` de la première fonction obligatoire.

En déclarant notre variable à cet endroit, nous en faisons une variable globale. C'est tout simplement une variable que l'on pourra lire et écrire depuis n'importe quel endroit du programme, donc depuis l'intérieur de n'importe quel bloc de fonction. Il ne faut pas le faire pour toutes les variables, mais c'est plus simple pour l'instant.

Déclarer la variable pour réserver de l'espace mémoire, c'est bien, mais le but est tout de même de l'utiliser. Comment faire pour stocker à l'emplacement symbolisé par cette variable la valeur que l'on va lire dans la boucle au moyen de la fonction `digitalRead()` ?

Souviens-toi de la fonction mathématique dont j'ai parlé plus haut. Lorsqu'une variable `x` vaut 6, la fonction `f(x)` vaut 8. Il en va de même pour les fonctions qui renvoient une valeur, ce qui est le cas de `digitalRead()`. Quand tu appelles cette fonction en écrivant son nom avec le numéro de la broche qu'elle doit lire, elle va exécuter toute une ribambelle d'instructions élémentaires puis renvoyer la valeur à l'endroit où se trouve le nom. C'est grâce à cette magie que nous pouvons écrire une instruction d'affectation qui consiste à copier une valeur dans une variable, comme ceci :

```
monNomDeVariable = nomFonction(paramètre);
```

Mettons cela en pratique immédiatement.

1. Pose le curseur clignotant à gauche du `v` du premier mot `void` du programme et appuie deux fois sur la touche Entrée pour insérer deux lignes.

2. Avec la flèche Haut du curseur, remonte à la toute première ligne, puis insère l'instruction de déclaration suivante :

   ```
   boolean bValeur ;
   ```

3. Descends maintenant avec la flèche Bas et les flèches Gauche et Droite (plus pratique qu'avec la souris) jusqu'à te placer à gauche du nom de la fonction `digitalRead(7)`.

4. Insère à cet endroit le nom de la variable que nous avons déclarée, une espace, un signe égale et une autre espace.

   Tu dois obtenir le même résultat que dans le listing suivant.

Listing 4.3 : Ajout d'une variable (CH04_Digit3)

```
/* ARDUIPROG_KIDZ
 *  CH04A_Digit3.ino
 *  OEN1703
 */

boolean bValeur;

void setup() {
  pinMode(7, INPUT);
```

```
}

void loop() {
  bValeur = digitalRead(7);
}
```

---

5. Contrôle ta saisie puis lance une vérification.
6. Téléverse ta nouvelle version du programme sur la carte.

   Que remarques-tu ? Rien de plus que dans la version précédente. Pourtant, dorénavant, la valeur renvoyée par la fonction de lecture est bien copiée dans notre variable, et elle se trouve donc dans la mémoire des variables. Il nous manque quelque chose.

Le problème est que nous sommes à l'extérieur du mikon, et nous n'avons pas accès à tout ce qui se passe dans ce carré de 3 mm. Pour le savoir, nous allons utiliser l'outil déjà rencontré dans le chapitre précédent, qui permet de demander au mikon de nous renvoyer des valeurs. Nous allons nous en servir pour faire renvoyer la valeur lue afin qu'elle soit affichée dans le Moniteur série.

Affichons la valeur dans le Moniteur série

L'outil Moniteur série fait partie de l'atelier Arduino, comme tu le sais. Pour pouvoir l'utiliser depuis ton programme, il faut d'abord effectuer un réglage de la vitesse à laquelle les données vont transiter entre la carte et la fenêtre du moniteur.

Comme pour le choix du mode d'utilisation de la broche numérique, nous allons ajouter un appel à une fonction qui ne doit

être réalisé qu'une fois. Cet appel va donc être inséré dans la première des deux fonctions obligatoires, `setup()`. La fonction que nous allons ajouter est d'un nouveau style. Voici comment doit s'écrire l'appel à cette fonction :

```
Serial.begin(9600);
```

Tu remarques d'abord le point dans le nom. C'est nouveau. En fait, le nom `Serial` (qui DOIT commencer par un S majuscule) est le nom d'un objet. Oui, tout à fait, tu fais déjà de la programmation orientée objets (un enrichissement du langage C nommé C++) ! Mais pas trop vite. Nous reviendrons sur les objets plus tard.

Pour l'instant, sache qu'un objet est quelque chose d'encore plus évolué qu'une simple fonction. Un objet « possède » des fonctions et des variables. Pour faire exécuter une fonction qui appartient à un objet, on doit indiquer d'abord le nom de l'objet propriétaire, puis un point, et le nom de la fonction.

Ici, la fonction porte le nom `begin()`, qui veut dire « débuter » . Il faut fournir un seul paramètre numérique, qui est la vitesse de communication, choisi parmi les valeurs qui sont proposées dans la liste de sélection du coin inférieur droit du moniteur (revois si nécessaire le chapitre précédent).

Il y a une seconde fonction à utiliser, et c'est la plus importante : celle qui va envoyer la valeur lue par le mikon vers l'atelier pour que celui-ci affiche la valeur dans le Moniteur série.

Cette autre fonction de l'objet `Serial` porte le nom `println()`, ce qui signifie « imprimer puis passer à la ligne suivante » (ln veut dire « ligne »). Voici comment s'écrit l'appel :

```
Serial.println(valeur);
```

Passons à la pratique.

1. Dans la fenêtre de l'éditeur, clique à droite du signe point-virgule de la seule fonction qui se trouve

actuellement dans le corps de la première fonction obligatoire `setup()` et appuie sur la touche Entrée pour créer une nouvelle ligne vide.

2. Tu remarques que le curseur de saisie est automatiquement placé à deux colonnes de décalage par rapport à la marge gauche. On appelle cela *l'indentation*. Le fait d'indenter ou pas n'a aucune importance pour le compilateur. Le programme sera considéré comme correct même si tu ne mets aucun espace de marge gauche.

   En revanche, cela fait une grosse différence de lisibilité du texte source. Il est très important de voir aisément à quel niveau tu te trouves dans les imbrications. Tu le constateras par toi-même dans les projets plus conséquents des chapitres suivants.

3. Saisis maintenant l'instruction suivante :

   ```
   Serial.begin(9600) ;
   ```

   N'appuie pas sur la touche Entrée, c'est inutile ici.

4. Procède de la même manière en te plaçant après le point-virgule de la seule instruction de l'autre fonction, `loop()`, et appuie sur la touche Entrée puis saisis la deuxième nouvelle instruction :

   ```
   Serial.println(bValeur) ;
   ```

La nouvelle version du programme est maintenant complète. Commence par vérifier qu'il n'y a pas d'erreur de saisie au moyen de la commande **Vérifier**.

Test du projet

Nous allons enfin recueillir les fruits de nos efforts.

1. Branche la carte si elle ne l'est pas encore.
2. Dans l'atelier, utilise le bouton Moniteur série en haut à droite pour ouvrir la fenêtre du moniteur.

La fenêtre refuse de s'ouvrir si la carte Arduino n'est pas connectée et détectée.

Vérifie bien que le fil volant ne touche rien par son extrémité libre.

3. Repositionne si nécessaire la fenêtre du Moniteur série pour qu'elle soit visible à côté de celle du texte source.
4. Dans la fenêtre du moniteur, vérifies en bas à droite que la vitesse de dialogue est bien la même que celle que tu demandes dans l'instruction `Serial.begin()`, soit 9600.

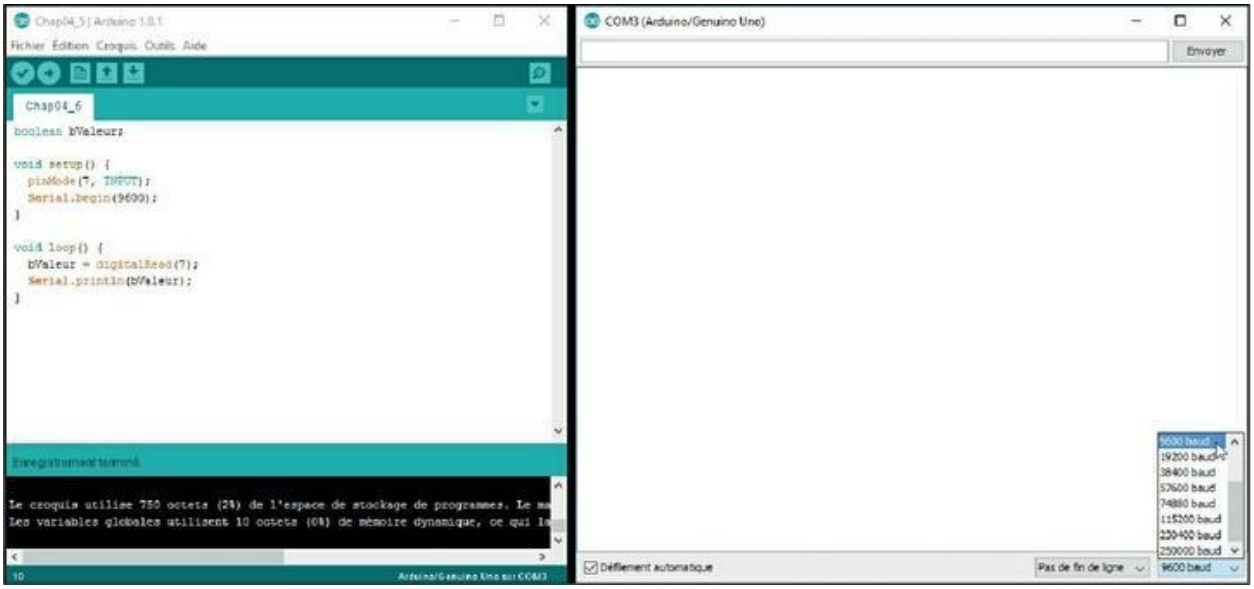


Figure 4.6 : Vérification de la vitesse de transmission.

Voici enfin le grand moment attendu. Utilise la commande **Téléverser** pour transférer ton projet et en lancer l'exécution. Tiens-toi prêt à bien observer ce qui va apparaître dans la fenêtre du Moniteur série.

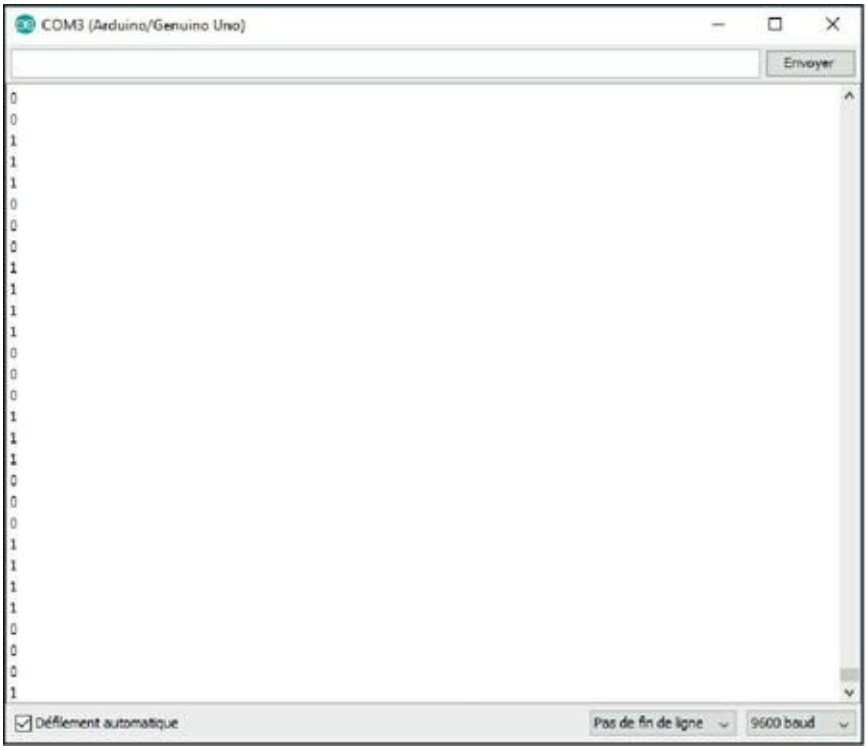


Figure 4.7 : Affichage des valeurs numériques.

Que constates-tu ? Tu vois défiler des 0 et des 1 sans cesse. C'est tout à fait normal : le fil sert d'antenne et récupère tous les parasites qui traînent dans l'air.

Dans le monde moderne, il y a de plus en plus d'ondes dans notre environnement ; ce sont ces ondes qui donnent cet affichage incohérent.

5. Prends entre deux doigts l'extrémité libre du strap et appuie la broche sur le boîtier métallique de la prise USB, puis maintiens-le appuyé. Observe l'affichage dans le Moniteur série. Normalement, tu ne dois voir que des 0. C'est parce que tu forces la valeur d'entrée à 0 V, car la prise USB, ou plutôt son embase, est reliée à la masse, c'est-à-dire à 0 V.

   Dès que tu relâches l'appui, la ronde des 0 et des 1 reprend.

Avec ce strap qui pendouille, l'entrée est flottante, et le mikon détecte des 0 et des 1 au fur et à mesure des parasites captés par le fil. Pour certains capteurs, et notamment pour les interrupteurs, il faudra éviter cet état fluctuant. Mais tout a été prévu.

Il suffit d'utiliser une variante du mot réservé qui sert à dire que nous voulons utiliser la broche en entrée. En effet, le mikon possède des petites résistances débrayables dans ses circuits. En activant une telle résistance, on oblige l'entrée à être reliée au +5 V, sauf lorsqu'on force le courant à rejoindre le 0V. Essayons cela immédiatement.

Nous en profitons pour découvrir une autre fonction prédéfinie qui sert à ralentir un peu l'affichage. C'est une fonction très simple qui se contente de faire patienter pendant le temps indiqué avant de reprendre la suite des opérations. La fonction se nomme fort logiquement `delay()`.

Voici donc les deux modifications que nous apportons au projet :

1. Dans le texte source, clique juste après la lettre T du mot INPUT dans la fonction de configuration.
2. Ajoute alors sans espace les lettres suivantes, en commençant par le caractère de soulignement :

   _PULLUP

Le résultat doit correspondre à l'instruction suivante :

```
pinMode(7, INPUT_PULLUP) ;
```

3. Pour ralentir un peu l'affichage, nous ajoutons l'appel de fonction suivant à la suite des deux qui existent déjà dans la fonction principale de la boucle :

```
delay(500) ;
```

La fonction `delay()` fait patienter pendant le nombre de millisecondes indiqué. Ici, nous demandons 500 ms, soit une demi-seconde.

Après activation de la résistance interne de tirage (PULLUP) et ajout de la pause, le projet doit avoir l'aspect suivant :

Listing 4.4 : Version finale du projet (CH04_Digit4)

```
/* ARDUIPROG_KIDZ
 *  CH04A_Digit4.ino
 *  OEN1703
 */

 boolean bValeur;

void setup() {
  pinMode(7, INPUT_PULLUP); // MODIF ICI
  Serial.begin(9600);
}

void loop() {
  bValeur = digitalRead(7);
```

```
  Serial.println(bValeur);
  delay(500);
}
```

Le qualificatif PULLUP que nous avons ajouté désigne ce mécanisme interne consistant à mettre en activité une résistance entre la tension positive de 5 V et la broche, ce qui force cette broche à l'état 1, sauf quand nous décidons de la forcer l'état 0. Mettons cela en pratique maintenant.

1. Une fois les modifications effectuées, sauvegarde ton programme si nécessaire avec la commande **Fichier/Enregistrer sous** puis demande son téléversement. Dorénavant, l'affichage est beaucoup plus calme et, surtout, tu ne vois apparaître que des 1.

2. Comme précédemment, fais contact entre la pointe libre du strap et le boîtier de la prise USB. Tu ne dois voir que des 0.

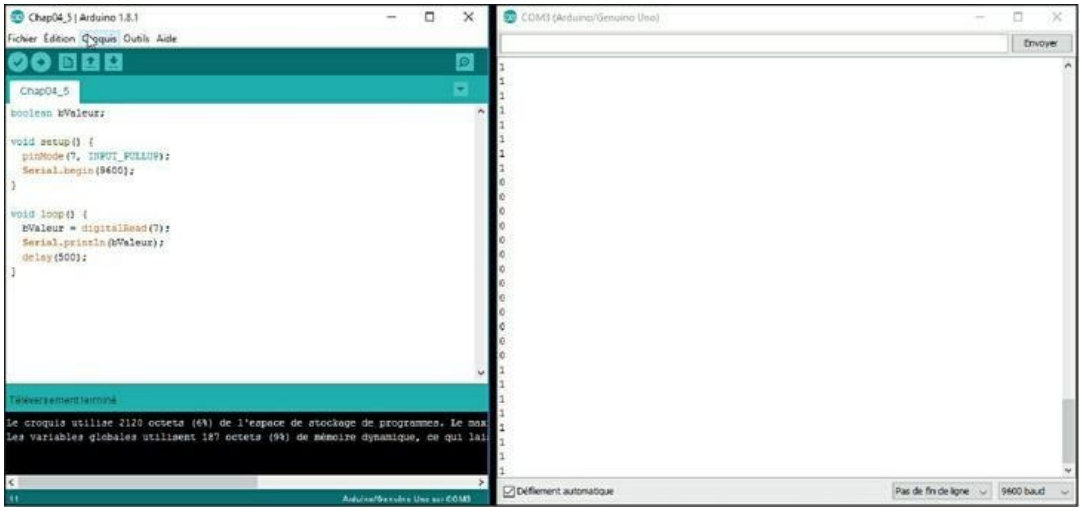


Figure 4.8 : Test de lecture d'une entrée stabilisée.

Nous savons maintenant comment exploiter une entrée numérique. Cette technique va te permettre d'exploiter toute une gamme de capteurs à sortie numérique, c'est-à-dire qui fournissent comme résultat de leurs mesures soit la valeur 0, soit la valeur 1.

Mais il existe aussi des capteurs analogiques qui fournissent comme résultat de leurs mesures une tension variant progressivement entre 0 V et 5 V. Si tu branches un tel capteur analogique sur une entrée numérique, tu n'obtiendras que les valeurs suivantes :

- Lorsque la tension fournie par le capteur se situera entre 0 V et 0,8 V environ, tu obtiendras un 0.
- Lorsque la tension sera supérieure à 2,8 V et jusqu'à 5 V, tu obtiendras un 1.
- Lorsque la tension sera dans la zone grise entre les deux, tu ne pourras jamais prévoir si tu obtiendras 0 ou 1.

Il faut donc un autre moyen pour récupérer une mesure analogique. C'est à cela que servent les broches analogiques du connecteur du bas de ta carte. Nous allons les découvrir dans le prochain chapitre.

Récapitulons

Dans ce chapitre, nous avons appris :

- à écrire des appels de fonctions prédéfinies ;
- à déclarer une variable ;
- à régler le Moniteur série et à afficher des valeurs ;
- à stabiliser une entrée numérique.

Chapitre 5
À l'écoute du monde réel

AU MENU DE CE CHAPITRE :

- **Lecture analogique**
- **Des types de données entiers**
- **Ta première fonction**
- **Une instruction conditionnelle**
- **Contrôler le format dans le Moniteur série**
- **Un projet de thermomètre**

Il y a une énorme différence entre le monde des ordinateurs et le monde réel, physique, le monde IRL (In Real Life) : dans le monde réel, il n'y a pas vraiment de nuit absolument noire, ni de jour absolument clair. Même par une nuit sans lune, on parvient encore à deviner des silhouettes. Même par un jour de plein soleil au beau milieu de l'été, nous ne sommes pas éblouis, et heureusement.nécessaire.

Le monde réel est celui du ni oui, ni non. Prenons l'exemple de la pesée des valises à l'aéroport. Les balances peuvent, par exemple, être construites pour savoir peser entre 1 g et 50 kg. À raison d'environ 1 milliard de voyageurs par an, donc autant de valises, si la mesure est suffisamment précise, il n'y aura jamais deux valises pesant exactement la même masse (on ne dit pas poids, mais masse).

Si on branche la balance sur l'entrée numérique d'un ordinateur, on pourra seulement savoir s'il y a une valise ou pas. Cela ne permettra

pas de calculer le montant du supplément à payer lorsqu'une valise pèse plus lourd que la limite autorisée.

En pratique, on fait des arrondis : on ne pèse pas les valises au microgramme près. L'arrondi, c'est un premier pas menant du continu du réel au discontinu de l'informatique.

Alors, comment faire pour que notre mikon sache exploiter une valeur qui n'est pas numérique ? Comment faire pour qu'il puisse exploiter l'un des nombreux types de capteurs qui sont conçus pour fournir une tension variant entre 0 V et 5 V, en passant par une quasi-infinité de valeurs intermédiaires ?

Observe ta carte Arduino. Tu remarques sur le connecteur en bas à droite de ta carte les six broches marquées A0 à A5 avec la mention ANALOG IN. C'est bien à ces broches-là que nous allons nous intéresser dans ce chapitre. Nous allons y brancher un équipement que va envoyer une tension variant progressivement. Mais tu sais qu'un processeur tel que le mikon est une machine binaire : elle ne peut accepter que des 0 et des 1. Au-dessous d'une certaine tension, ce sera 0, et au-dessus d'une autre, ce sera 1. Il ne fait pas dans la nuance ! Il y a donc une astuce.

Cette astuce se nomme CAN, acronyme pour convertisseur analogique-numérique (en anglais, DAC pour Digital-Analog Converter).

Le principe du CAN est de comparer la tension sur la broche d'entrée à une tension qu'il génère en interne. Il fonctionne sur le même principe que le jeu consistant à deviner un nombre. Rappelons-le.

Supposons que je choisisse un nombre entre 1 et 100 et que je te demande de le deviner. Pour chaque essai, je te dis si le nombre est supérieur ou pas. Quelle est la méthode la plus efficace ? Tu vas d'abord proposer 50. Si c'est supérieur à 50, tu vas dire 75. À chaque étape, tu vises la moitié de la plage de possibilités restantes. Le CAN fonctionne exactement de la même manière. Si la plage de tensions possibles va de 0 V à 5 V, le CAN va commencer par comparer la tension à mesurer à 2,5 V. Si elle est inférieure, il va

réessayer avec 1,25 V. Il procède ainsi de suite jusqu'à trouver la valeur numérique la plus proche.

Le CAN du mikon AT328 qui équipe ta carte Arduino offre une précision de 10 bits, ce qui signifie qu'il peut distinguer 1024 niveaux différents. Nous verrons dans le chapitre suivant que la valeur 1024 peut être codée sur 10 bits (c'est le double de 512 codé sur 9 bits, lui-même le double de l'octet codé sur 8 bits).

Il existe de plus en plus de capteurs intelligents qui réalisent eux-mêmes la conversion de l'analogique vers le numérique. Ils transmettent donc directement des valeurs binaires en utilisant une interface nommée I2C. Nous verrons cela plus tard.

Le projet Analo

Passons au montage d'un circuit permettant de vérifier le fonctionnement du convertisseur CAN. Ce montage, comme celui du chapitre précédent, sera extrêmement simple puisqu'il se résumera à un fil volant branché sur une broche d'entrée.

1. Vérifie que ta carte n'est pas branchée, puis enlève tous les straps qui sont éventuellement présents et branche un seul strap sur la broche analogique A0.
2. Vérifie que l'extrémité libre du strap ne touche à rien et pend dans le vide.
3. Mets la carte sous tension en connectant le câble USB à l'ordinateur.

   C'est tout. Nous pouvons maintenant passer à la rédaction du texte source.

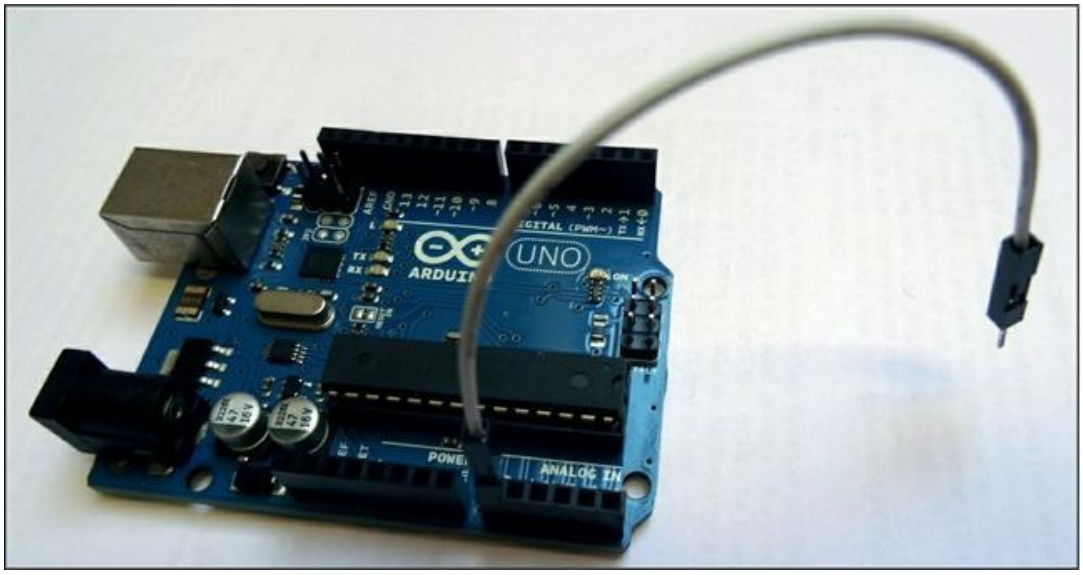


Figure 5.1 : Aspect du montage pour un test d'entrée analogique.

Conception du programme

Ce projet ressemble beaucoup à celui du chapitre précédent. La différence essentielle est que nous allons avoir besoin d'une autre fonction prédéfinie. Au lieu de `digitalRead()`, nous allons utiliser `analogRead()`. C'est une fonction standard disponible dès le départ dans l'atelier Arduino. Il suffit d'indiquer son nom, et elle sera reconnue.

L'autre différence se situe au niveau de la variable dans laquelle nous allons récupérer la valeur numérique issue du travail du convertisseur analogique numérique. J'ai dit plus haut que ce convertisseur savait générer une valeur entre 0 et 1023.

Le système de comptage humain est décimal. Pour écrire la valeur 999, trois chiffres suffisent, mais pour écrire 1000, il en faut quatre. Dans la numérotation binaire des ordinateurs, il faut dix bits à 0 ou à 1 pour représenter la valeur décimale 1023. Nous y reviendrons dans le prochain chapitre, mais sache déjà que 1023 décimal s'écrit `11 11111111` en binaire. Si tu comptes bien, il y a dix chiffres 1, c'est-à-dire dix bits.

Notre mikon, comme tous les processeurs existant actuellement dans le monde, travaille avec des paquets de 8 bits au minimum, des octets. Autrement dit, pour stocker la valeur 1023, il nous faut deux octets. Dans le langage C de notre mikon, cette largeur de données correspond au type de données portant le nom int.

Le mot réservé int est l'abréviation de integer qui signifie entier. Les valeurs entières sont celles qui n'ont pas de virgule. Les nombres 37 ou 655 sont des valeurs entières, mais pas 1,25 ni 36,99. Note au passage que, pour séparer les chiffres des unités des chiffres après la virgule dans tes programmes, tu ne dois plus utiliser la virgule, mais le point. Tu écriras donc 1.25 et 36.99.

Avant de saisir le texte source, je te propose de bien préparer tes conditions de travail, comme ceci :

1. Si l'atelier IDE n'est pas démarré, fais-le.
2. Utilise la commande **Fichier/Nouveau** pour lancer un nouveau projet.
3. Ferme l'autre fenêtre qui a été ouverte au démarrage.
4. Prends l'habitude d'attribuer immédiatement un nom à ton projet en utilisant la commande **Fichier/Enregistrer**. Choisis ce nom avec soin. J'ai par exemple opté pour le nom suivant pour la première version du projet de ce chapitre :

   `CH05_Analo1`

   Ne fais pas commencer le nom du projet par un chiffre. L'atelier n'aime pas cela.

5. Une fois que tu es prêt, procède à la saisie des lignes du listing suivant.

Listing 5.1 : Texte source de la version 1 de Analo

```
/* ARDUIPROG_KIDZ
 *  CH05A_Analo1.ino
 *  OEN1703
 */

int broLecture = 14;  // Broche A0 = 14
int iValLue = 0;

void setup() {
  pinMode(broLecture, INPUT);
  Serial.begin(9600);
}
void loop() {
  iValLue = analogRead(broLecture);
  Serial.println(iValLue);
  delay(500);
}
```

Remarques de lecture

Tu constates que je commence par déclarer une première variable nommée `broLecture` en lui donnant la valeur initiale 14. Cela permet d'utiliser le nom de variable partout où il faudrait indiquer le numéro de la broche analogique. Tu remarques au passage que la première des broches analogiques prend le numéro suivant après celui de la broche numérique 13. Autrement dit, les broches analogiques A0 à A5 portent les numéros 14 à 19.

La deuxième variable globale que je déclare porte le nom `iValLue` Elle est aussi déclarée de type `int`. J'ai expliqué plus haut

pourquoi il nous fallait ce type de variable.

Nous retrouvons ensuite nos deux fonctions obligatoires. Dans la fonction de préparation `setup()`, nous réglons notre broche d'entrée en mode entrée (INPUT). Cela peut paraître superflu : une broche analogique d'entrée est une broche d'entrée, non ? Eh bien, non : malgré leur nom, ces six broches peuvent aussi être utilisées en sortie. J'en profite pour te prévenir que les broches de sortie sont toujours numériques, et qu'il faudra une autre astuce pour qu'elles puissent simuler une tension analogique. Nous verrons cela dans le prochain chapitre.

La deuxième instruction de la fonction `setup()` est déjà connue. Elle configure la vitesse de dialogue entre la carte et la fenêtre du Moniteur série.

Dans notre fonction principale, nous commençons par stocker dans notre variable la valeur qui aura été lue sur la broche d'entrée par notre nouvelle fonction `analogRead()`. Rappelons cette première ligne pour la décomposer :

```
iValLue = analogRead(broLecture);
```

Il y a ici deux opérations dans une seule ligne :

- D'abord, nous demandons l'exécution de la fonction `analogRead()`. Quand son exécution est terminée, cette fonction va renvoyer une valeur ; cette valeur va d'une certaine façon surgir à la place de l'appel à la fonction.
- La deuxième opération consiste simplement à recopier cette valeur renvoyée dans l'emplacement mémoire qui est symbolisé par le nom de la variable.

Les deux instructions suivantes sont les mêmes que dans le chapitre précédent. Nous affichons la valeur que nous venons de lire avec un saut de ligne de l'affichage (grâce à la variante

`Serial.println()` au lieu de `Serial. print()`) puis nous provoquons une petite pause d'une demi-seconde (500 millisecondes).

Passons aux tests !

Test du projet Analo en version 1

1. Utilise la commande **Téléverser** pour transférer le programme après compilation.

   S'il y a une erreur, relis bien le texte source en vérifiant notamment les accolades, les signes point-virgule et les parenthèses.

2. Au bout de deux ou trois secondes, le programme doit avoir commencé son exécution. Ouvre la fenêtre du Moniteur série. Qu'observes-tu ?

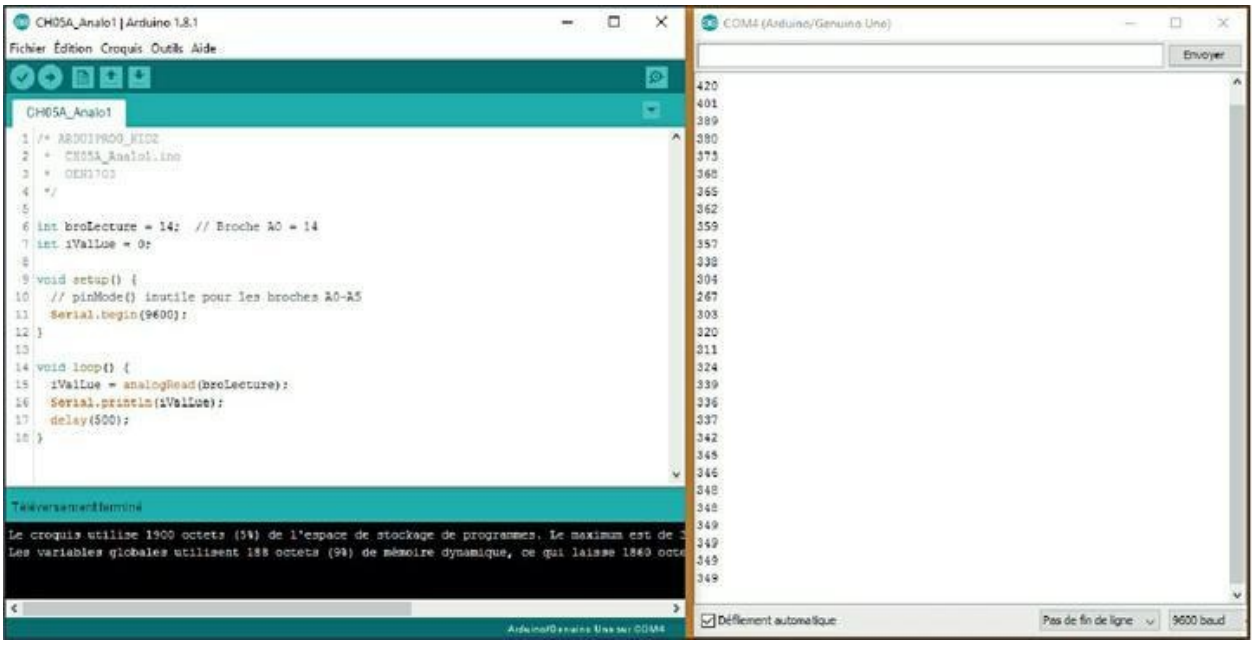


Figure 5.2 : Test de la première version du projet Analo.

On voit apparaître des valeurs qui se situent entre 200 et 1000 environ. La valeur fluctue en fonction des parasites de ton environnement réel. Tu peux par exemple approcher ton téléphone portable de la carte, ce qui aura certainement une influence.

3. Prends le bout libre du strap et fais-le toucher le boîtier de la prise USB sur la carte.

   L'affichage devrait passer à 0 puisque tu présentes une tension de 0 V sur la broche d'entrée analogique.

4. Touche la pointe du strap avec un doigt pour voir si cela change la valeur.

Ce test permet de confirmer que le mikon sait détecter une tension variable en la transformant en une valeur entre 0 et la limite de 1023.

Nous allons utiliser cette possibilité dans le second projet de ce chapitre, mais auparavant, progressons dans l'apprentissage des techniques de programmation, puisque c'est le but principal de ce livre, apprendre à programmer !

La prochaine étape va être d'une importance capitale : tu vas apprendre à déclarer et exploiter ta première fonction.

Ta première fonction

Combien de fonctions prédéfinies avons-nous déjà rencontrées au cours du chapitre précédent et du début de celui-ci ? Environ une dizaine.

Il y a tout d'abord les deux fonctions obligatoires `setup()` et `loop()`. Elles sont entièrement transparentes pour toi, dans le sens où tu as accès à leur contenu, puisque c'est toi qui écris les lignes entre les deux accolades de chacune.

Il y a ensuite les fonctions dont tu as utilisé le nom pour profiter du service correspondant. C'est le cas de `digitalRead()`, `delay()`, `pinMode()`, `Serial. print()` et de sa variante `println()`, et enfin de `analogRead()`. Toutes ces fonctions prédéfinies constituent des boîtes noires, puisque tu ne connais pas leur contenu exact. Il te suffit de savoir comment les utiliser (les invoquer) et quel profit tu peux en tirer.

Il n'est pas interdit de connaître le contenu des fonctions prédéfinies. Elles sont toutes fournies selon le mode open source ; le texte source est donc accessible, mais tu verras cela plus tard.

Chacune des fonctions prédéfinies contient entre une dizaine et plusieurs centaines d'instructions. Tu constates dans l'éditeur que les fonctions reconnues par l'atelier sont affichées en orange. Dès que tu fais une faute de frappe dans le nom d'une fonction prédéfinie, tu le vois immédiatement parce qu'elle ne s'affiche pas en orange.

Les deux fonctions obligatoires s'affichent dans une couleur spéciale.

Dans le langage de programmation C de l'Arduino, il y a une soixantaine de fonctions prédéfinies. Tu pourras y ajouter en fonction de tes besoins un nombre non limité de fonctions supplémentaires, réunies par groupes logiques dans des librairies.

Le point essentiel est qu'il ne faut jamais utiliser le même nom pour deux choses différentes, comme dans la vie courante.

Ceux qui apprennent la programmation dans une autre langue que l'anglais ont un avantage à ce niveau : les identifiants qu'ils ont envie d'utiliser pour leurs fonctions et leurs variables risquent rarement d'entrer en conflit avec des mots réservés du langage, ces derniers étant toujours en anglais.

En plus des fonctions, nous avons utilisé quelques mots réservés du langage, qui s'affichent en bleu dans l'éditeur :

- les trois types de données boolean, int et byte ;
- les mots INPUT et INPUT_PULLUP ;

- un certain nombre de signes de ponctuation (parenthèses, accolades, etc.).

Nous avons aussi créé trois identifiants pour trois variables différentes : `bValeur`, `iValLue` et `broLecture`.

Nous allons donc choisir un joli nom pour notre première fonction. Mais petite question préalable : pourquoi créer des fonctions ?

On travaille mieux à plusieurs

En créant des fonctions, on répartit le travail dans plusieurs blocs, ce qui offre de nombreux avantages :

- Le programme est plus facile à lire et à comprendre.
- Le programme est plus facile à mettre au point, c'est-à-dire à déboguer.
- Une fonction peut facilement être copiée et réutilisée dans un autre projet si elle correspond aux mêmes besoins.

Bref, en créant des fonctions, tu prends la bonne habitude de modulariser.

En guise de première fonction, je te propose d'extraire de la fonction principale `loop()` les deux lignes d'instructions qui effectuent la lecture de la prochaine valeur puis l'affichage de cette valeur.

Puisque l'heure est venue de choisir le nom de notre nouvelle création, voici les règles à satisfaire :

- Ton nom ne doit évidemment pas entrer en conflit avec aucun des noms déjà connus par le compilateur, donc aucun des noms des fonctions internes ni aucun

> mot réservé. Bien sûr, tu ne connais pas encore tous les mots réservés et noms de fonctions, mais tu verras qu'il y a peu de chances de tomber sur un homonyme en français.
>
> » Ensuite, il faut prendre l'habitude de ne jamais utiliser de lettres accentuées dans les noms. Lorsque tu as besoin de simuler un espace, utilise le caractère de soulignement (_). Je te conseille aussi d'adopter l'écriture en dos de chameau, comme ceci :
>
> ```
> int droMaDaire
> ```
>
> » Enfin, une bonne pratique consiste à toujours utiliser un verbe d'action pour les fonctions, ce qui est normal puisque les fonctions agissent. Tu réserves les noms statiques à tes variables.

Passons donc au codage.

Déclaration d'une fonction

1. L'atelier doit être ouvert avec la première version du projet en cours, mais le nouveau nom sera enregistré *via* la commande **Enregistrer sous**.
2. Sélectionne les deux lignes suivantes et utilise la commande **Édition/Couper**.

```
iValLue = analogRead(broLecture) ;
Serial.println(iValLue) ;
```

3. Descends tout à la fin du texte source, insère une ligne d'espace avec Entrée puis colle les deux lignes (**Édition/Coller**).

4. Place-toi au début de la première des deux nouvelles lignes, celle qui commence par `iValLue`, et insère une ligne vide avec la touche Entrée.

5. Nous allons insérer ici la ligne de tête de la fonction. Voici d'abord la syntaxe formelle de la tête d'une fonction :

   ```
   nomType nomFonction() {
   ```

   Comme c'est notre première fonction, nous allons rester modestes : notre fonction ne renverra pas de valeur, ce que nous décidons en commençant par le mot clé `void`.

   Pour simplifier encore, la fonction ne va pas attendre de valeur en entrée, c'est-à-dire une ou plusieurs valeurs qui sont transmises à la fonction quand elle démarre et permettent de faire varier son comportement. On appelle cela des *paramètres d'entrée*, et c'est ce que l'on indique entre les deux parenthèses après le nom de la fonction. Ici, nous laissons le couple de parenthèses vide.

   La ligne de tête va donc s'écrire ainsi si nous choisissons comme nom de fonction `lireAna()` :

   ```
   void lireAna() {
   ```

Tu peux constater que cette ligne ne se termine PAS par un point-virgule.

6. Place-toi enfin tout à la fin de ton texte source, donc après le dernier signe point-virgule. Insère une ligne vide puis saisis une accolade fermante contre la marge gauche.

   ```
   }
   ```

   C'est l'accolade qui ferme le bloc qui a été ouvert par l'accolade ouvrante à la fin de la ligne de tête de notre fonction.

   Si nécessaire, enlève les espaces pour que cette accolade soit bien contre la marge gauche. Lorsque tu vas créer des sous-blocs, tu pourras ainsi facilement distinguer les différents niveaux par des décalages (indentations) de deux espaces par niveau.

7. Lance une vérification. La définition de la nouvelle fonction est terminée. En cas d'erreur, relis-toi bien.

8. Branche la carte et téléverse le projet. Il ne se passe plus rien, même pas besoin d'ouvrir le moniteur pour s'en assurer. Pourquoi ?

   Parce que nous ne demandons jamais à notre nouvelle fonction de travailler ! Il faut l'appeler dans la fonction principale `loop()`.

Des fonctions à l'appel

Pour écrire un appel à ta fonction, tu procèdes exactement comme pour une fonction prédéfinie : tu cites son nom, et comme c'est une instruction, tu ajoutes un signe point-virgule. C'est tout.

1. Remonte dans la fonction principale `loop()` et place-toi juste après l'accolade ouvrante, puis insère une ligne vide. L'éditeur devine que tu veux ajouter quelque chose dans le bloc de la fonction : il a automatiquement inséré deux espaces par rapport à la marge gauche. Saisis alors le nom que tu as choisi pour la nouvelle fonction, dans mon exemple, `lireAna()`, sans oublier la paire de parenthèses vide et le signe point-virgule qui doit marquer la fin de chaque instruction.

Rares sont les lignes qui ne se terminent pas par ce signe.

2. Il ne te reste plus qu'à vérifier tes travaux d'édition en comparant avec le listing complet suivant.

Listing 5.2 : Version 2 du projet Analo

```
/* ARDUIPROG_KIDZ
 *  CH05A_Analo2.ino
 *  OEN1703
 */

int broLecture = 14;  // Broche A0 = 14
int iValLue = 0;       //

void setup() {
```

```
  pinMode(broLecture, INPUT);
  Serial.begin(9600);
}
void loop() {
  lireAna();                    // MODIF
  delay(500);
}

void lireAna() {              // NOUVELLE
FONCTION
  iValLue = analogRead(14);
  Serial.println(iValLue);
}
```

Testons notre fonction

Tu as certainement très envie de voir ta nouvelle fonction fonctionner !

1. Commence par une petite vérification avec la commande Vérifier. Lis bien les messages qui apparaissent dans le panneau inférieur. S'il n'y a pas d'orange, tout va bien.

   S'il y a quoi que ce soit en orange, revérifie tout en t'aidant du listing précédent.

2. Tu peux maintenant rebrancher la carte Arduino si tu l'avais débranchée et lancer un téléversement.

   Ouvre le Moniteur série et observe le résultat. Il est strictement identique à la première version.

Tu disposes dorénavant d'une fonction que tu peux copier-coller dans un autre projet telle quelle ou en la modifiant ensuite. Ton programme est devenu modulaire.

Voyons maintenant comment faire varier le fonctionnement du programme en découvrant les instructions conditionnelles.

Une instruction conditionnelle

Dans la vie courante, nous faisons sans cesse des choix en fonction des conditions que nous constatons autour de nous. Quand tu te lèves le matin, tu regardes instinctivement tes pieds au moment de mettre tes chaussettes. Autrement dit :

```
SI pasDeChaussettes ALORS mettreChaussettes ;
```

Si, en regardant tes pieds, tu constates qu'il y a déjà des chaussettes autour, tu n'exécutes pas l'instruction `mettreChaussettes()`.
Nous allons réutiliser ce raisonnement pour ajouter une instruction conditionnelle de manière à ne passer à la ligne suivante qu'au bout de quelques affichages de valeurs.

Nous allons ajouter un bloc conditionnel dans notre fonction principale. Dans ce bloc, qui sera délimité par une paire d'accolades (comme tout bloc qui se respecte), nous allons insérer deux instructions : une pour provoquer un saut de ligne et une autre pour remettre à zéro le compteur au bout de 11 affichages.

Le bloc conditionnel que nous allons utiliser se fonde sur le mot réservé anglais if qui veut dire « si » . Voici sa syntaxe formelle :

```
if (condition_vraie) {
  première instruction;
  deuxième instruction;
  etc;
}
```

Si tu veux garder une version de chaque étape de ce projet, commence par enregistrer le fichier en cours sous un nouveau nom, pour montrer que c'est la troisième étape.

Voici les différentes actions d'édition que nous allons réaliser :

1. En début de texte source, nous allons déclarer une nouvelle variable pour notre compteur. Je te propose de choisir un nouveau type de données pour cette variable, le type byte :

   ```
   byte bCompteur = 0 ;
   ```

   Pourquoi ? Puisque nous voulons compter de 0 jusqu'à 10, il est inutile de gâcher de la mémoire avec une variable qui permettrait de compter jusqu'à 32 767, ce qui est le cas d'une variable du type int. Nous pouvons nous contenter (et largement) d'une variable de taille inférieure, qui n'occupe qu'un seul octet. Justement, le type `byte` permet de stocker une valeur entre 0 et 255.

   Après cette déclaration de variable, nous allons insérer un bloc conditionnel dans la fonction principale.

   Place-toi à la fin de la ligne qui constitue l'appel à ta nouvelle fonction et appuie sur la touche Entrée.

2. Dans cette nouvelle ligne, à deux espaces du bord gauche, insère la ligne de tête du nouveau bloc conditionnel :

   ```
   if (bCompteur > 10) {
   ```

Tu notes qu'il n'y pas de point-virgule ici. Ce qui est entre parenthèses est une *expression*. C'est le mot de passe pour entrer dans le bloc. Si cette expression est vraie, on passe. Donc, si la variable vaut plus de 10, c'est vrai. Tant qu'elle vaut entre 0 et 10 (compris), le test est faux et les deux instructions conditionnelles sont ignorées. Seule la pause est marquée puis on fait un nouveau tour de boucle principale.

3. Insère ensuite une nouvelle ligne dans laquelle tu places l'instruction qui provoque un saut de ligne dans le Moniteur série :

```
Serial.println("");
```

Le couple de guillemets vide n'est pas obligatoire, mais il permet d'insérer facilement un texte plus tard.

4. Utilise à nouveau la touche Entrée pour insérer la ligne suivante afin de remettre à 0 le compteur une fois qu'il a atteint la valeur maximale :

```
bCompteur = 0 ;
```

Il ne reste plus qu'à insérer une ligne de fermeture du bloc conditionnel. Cette accolade fermante doit se trouver à deux espaces du bord gauche pour qu'on la distingue bien de l'accolade de la fin de la fonction juste au-dessous.

```
}
```

Vérifie que ton projet ressemble au début de cette nouvelle version dans le listing suivant.

Listing 5.3 : Début de la version 3 de Analo

```
/* ARDUIPROG_KIDZ
 *  CH05A_Analo3.ino
 *  OEN1703
 */

int broLecture = 14;  // Broche A0 = 14
int iValLue = 0;
byte bCompteur = 0;   // AJOUT V3

void setup() {
  pinMode(broLecture, INPUT);
  Serial.begin(9600);
}

void loop() {
  lireAna();
  if (bCompteur > 10) {    // AJOUT V3
    Serial.println("");    // AJOUT V3
    bCompteur = 0;          // AJOUT V3
  }                         // AJOUT V3
  delay(500);
}
```

Pour l'instant, nous avons inséré un bloc contenant deux instructions qui ne seront exécutées que si la valeur de la variable `bCompteur` est supérieure à 10. Puisqu'elle vaut 0 au départ, cela n'arrivera jamais, à moins de la faire évoluer. C'est ce que nous

allons faire dans notre fonction qui va augmenter de 1 la valeur du compteur à chaque tour, donc chaque fois qu'elle sera appelée.

Descends dans la fonction `lireAna()` et place-toi à la fin de la dernière instruction pour insérer les deux lignes portant le commentaire // AJOUT V3 :

Listing 5.4 : Fin de la version 3 de Analo

```
void lireAna() {
  iValLue = analogRead(broLecture);
  Serial.print(iValLue);                    //
MODIF V3
  Serial.print(" ");                        //
AJOUT V3
  bCompteur++;                              //
AJOUT V3
}
```

La première des deux lignes sert à aérer l'affichage afin que les valeurs ne se collent pas les unes aux autres. Elle se contente d'afficher un espace.

C'est la seconde nouvelle ligne qui nous intéresse :

```
bCompteur++;
```

Ce double signe ++ collé à la fin du nom de la variable est une version abrégée pour augmenter la valeur de la variable d'une unité. On appelle cela l'incrémentation. La variable qui vaut 0 au départ finira ainsi par avoir la valeur 11, ce qui provoquera enfin l'exécution du bloc conditionnel.

Pour diminuer de 1 une valeur, on utilise le signe de soustraction sur le même principe :

`bCompteur--`

Tu as peut-être remarqué une modification que je n'ai pas encore mentionnée dans la nouvelle fonction : la première utilisation de la fonction d'affichage n'est plus `Serial.println()`, mais `Serial.print()`. En effet, nous ne voulons pas provoquer de saut de ligne après chaque affichage.

La Figure 5.3 montre l'aspect du programme complet dans sa troisième version.

Tu remarques les numéros de lignes en marge gauche. Ces numéros ne sont absolument pas ajoutés au texte source ; ils ne servent qu'au repérage dans l'affichage. Pour rendre ces numéros visibles, va dans les préférences de l'atelier (Figure 5.4). Sous Windows et Linux, c'est dans le menu **Fichier**. Sous MacOS, les préférences sont dans le menu **Arduino**.

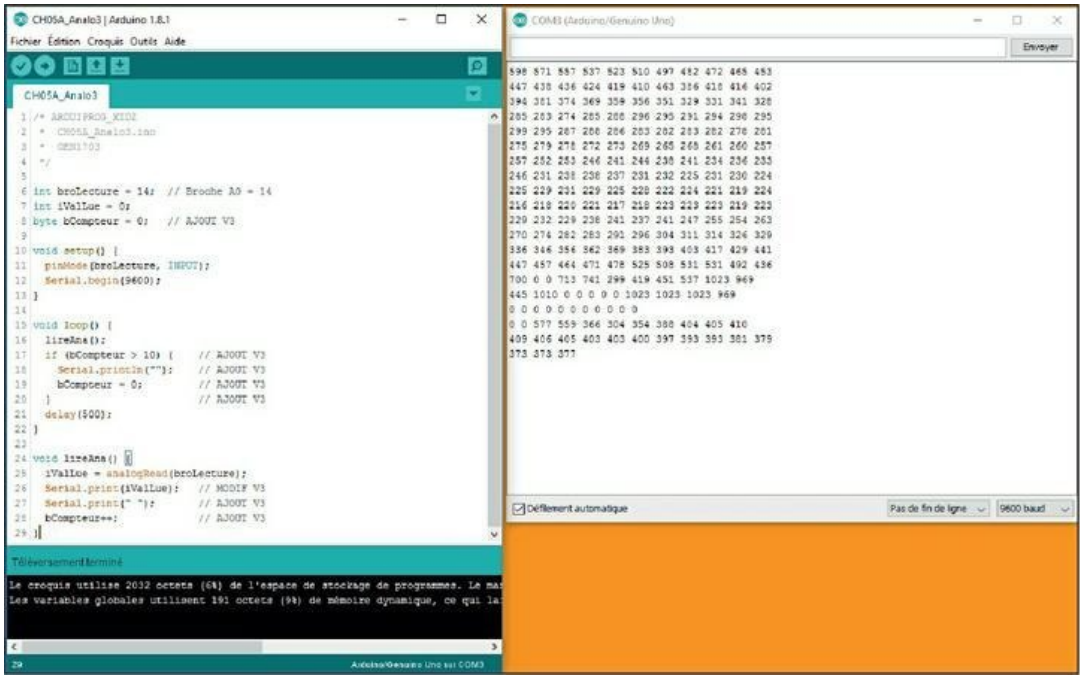


Figure 5.3 : Édition et exécution de Analo 3.

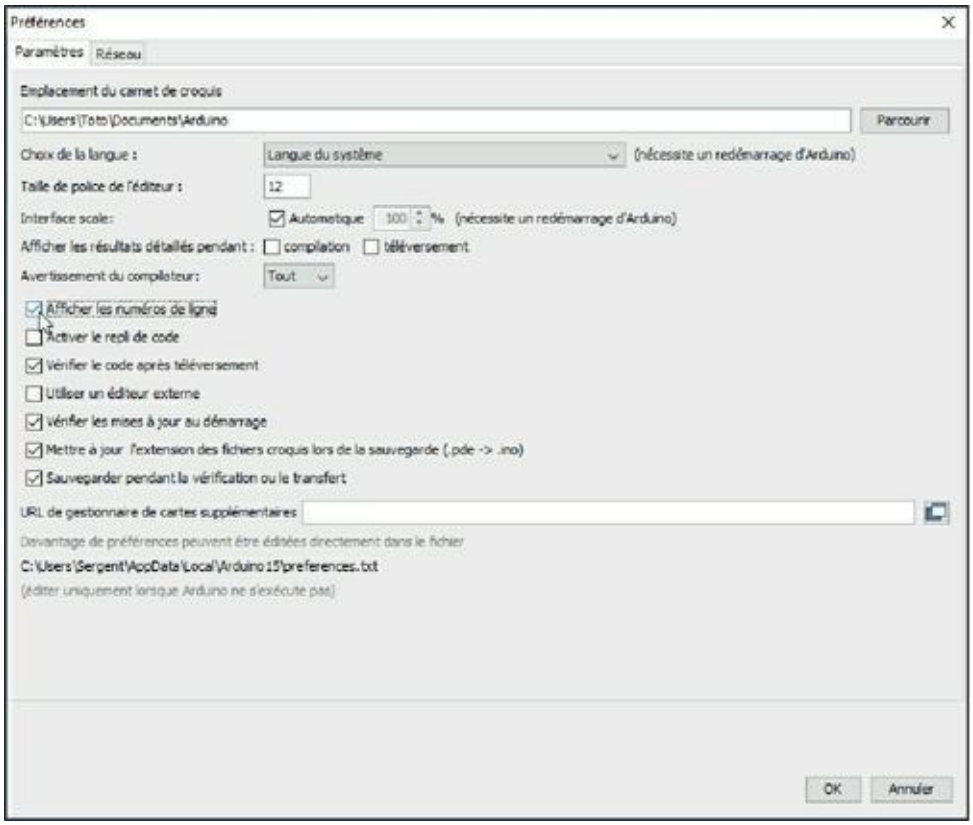


Figure 5.4 : L'option d'affichage des numéros de ligne dans les préférences.

Si plus tard, tu décides d'utiliser directement les outils de compilation sur la ligne de commande, il faut savoir qu'un bogue bizarre peut se produire parce qu'il manque un saut de ligne après la dernière accolade du programme. Tu peux donc en ajouter une ; cela ne change rien au résultat ici.

Testons la version 3

Je n'ai plus besoin de t'expliquer comment faire : d'abord **Vérifier**, puis **Téléverser** après avoir branché si nécessaire la carte. Tu ouvres ensuite le Moniteur série et tu regardes le résultat. N'hésite pas à toucher le bout du fil libre pour faire varier la valeur.

Tu remarques que l'affichage est beaucoup plus compact. C'était le but.

Compte les valeurs sur chaque ligne : il y en a bien 11.

Puisque nous sommes dans la découverte des fonctions, rendons celle que nous venons de définir un peu plus intelligente : nous allons lui faire renvoyer la valeur du compteur.

Renvoyons nos valeurs

Tu sais maintenant qu'il est possible de faire renvoyer une valeur par une fonction lorsqu'elle termine son exécution.

À vrai dire, on peut en faire renvoyer plusieurs, mais il faut dans ce cas les stocker dans un tableau, ce que nous verrons dans un autre chapitre.

Lorsqu'une fonction doit renvoyer une valeur, il faut tout d'abord indiquer quel sera le type de la valeur, ce qui se fait en remplaçant le mot clé void, dans la ligne de tête de la fonction, par le nom du type désiré. Dans notre exemple, nous voulons renvoyer le compteur de tours, et cette variable est de type byte. Cela nous conviendra.

Par ailleurs, si l'on déclare que l'on va envoyer une valeur, il faut tenir ses promesses. Dans le cas d'une fonction, cela suppose d'ajouter une instruction return avec, entre parenthèses, le nom de la variable dont on veut renvoyer la valeur. Je crois que tu es maintenant assez entraîné pour comprendre comment appliquer ces évolutions à la prochaine et dernière version de notre projet simplement en lisant le texte source suivant.

Listing 5.5 : Version 4 du projet Analo

```
/* ARDUIPROG_KIDZ
 *  CH05A_Analo4.ino
 *  OEN1703
 */
```

```
byte broLecture = 14;
int  iValLue    = 0;
byte bCompteur  = 0;            // AJOUT

void setup() {
  pinMode(broLecture, INPUT);
  Serial.begin(9600);
}

void loop() {                   // SUIVANTE
SUPPR
  if (lireAna() > 10) {         // MODIF
    Serial.println("");
    bCompteur = 0;
  }
  delay(500);
}

byte lireAna() {                // MODIF
  iValLue = analogRead(broLecture);
  Serial.print(iValLue);
  Serial.print(" ");
  return (bCompteur++);         // MODIF
}
```

Trois lignes ont changé et une ligne a disparu :

» D'abord dans notre fonction `lireAna()` : j'ai remplacé le mot void par le mot byte dans la ligne de tête.

» Dans la dernière ligne qui augmente de 1 la valeur de `bCompteur`, j'ai ajouté l'instruction return et j'ai enchâssé l'instruction `bCompteur++` dans les parenthèses de return. Autrement dit, l'effet devrait être le renvoi à l'endroit où nous appelons la fonction de la valeur de `bCompteur` une fois que celle-ci a été augmentée de 1. Nous allons voir que ce n'est pas exactement ce qui se passe en réalité.

» Dans la fonction principale `loop()`, nous avons supprimé la ligne de l'appel à la fonction. Comment ? Pas d'appel, pas d'affichage ? En fait, nous avons déplacé cet appel à la fonction en l'insérant dans la condition du test :

```
if (lireAna() > 10) {
```

Dorénavant, trois opérations successives ont lieu dans cette simple ligne :

1. La première action est l'exécution de la fonction `lireAna()`.
2. Au retour de la fonction, la valeur qu'elle renvoie apparaît « à la place » du nom de la fonction si on peut dire.
3. Nous pouvons enfin évaluer l'expression de test qui est devenue ceci :

```
EST-CE QUE ValeurRenvoyée EST SUPERIEURE A
10 ?
```

> Le bloc conditionnel n'est exécuté que si la réponse à cette question est oui, donc dès que cette valeur est égale à 11. Dans les autres cas, tout le bloc est ignoré et on ne fait que la pause d'une demi-seconde. Mais rassure-toi : l'appel à la fonction est toujours effectué. (Comment pourrait-on faire le test sinon ?)

Réalise ces quelques aménagements, puis vérifie et téléverse le programme afin de le tester. Observe bien l'affichage dans la fenêtre du moniteur et compte le nombre de valeurs affichées avant le saut de ligne.

Il y a un bogue : 12 valeurs sont affichées.

Préfixe ou suffixe, ce n'est pas pareil !

Nous voici face à notre premier bogue de logique. Le programme est considéré comme parfaitement correct pour le compilateur, la preuve en étant qu'il accepte de le téléverser vers la carte. Pourtant, nous faisons 12 tours de boucle avant de changer de ligne. Qu'est-ce qu'il se passe ?

En fait, il faut se méfier des notations abrégées pour incrémenter ou décrémenter une variable. Lorsque tu écris `variable++`, la valeur que possède la variable est utilisée d'abord, puis elle est augmentée. Pour qu'elle soit modifiée avant d'être utilisée, il faut écrire `++variable`. Subtil, non ?

Procède maintenant à cette modification en écrivant de la manière suivante la dernière ligne puis en téléversant pour vérifier qu'il n'y a plus que 11 tours de boucle, comme nous le voulons :

```
return (++bCompteur) ;
```

Puisque le projet fonctionne comme prévu, nous allons pouvoir passer à autre chose : apprendre à utiliser un premier véritable composant capteur. Je te propose un capteur de température afin de savoir s'il fait chaud dans la pièce où tu te trouves. Ce sera également notre premier contact avec une plaque d'expérimentation.

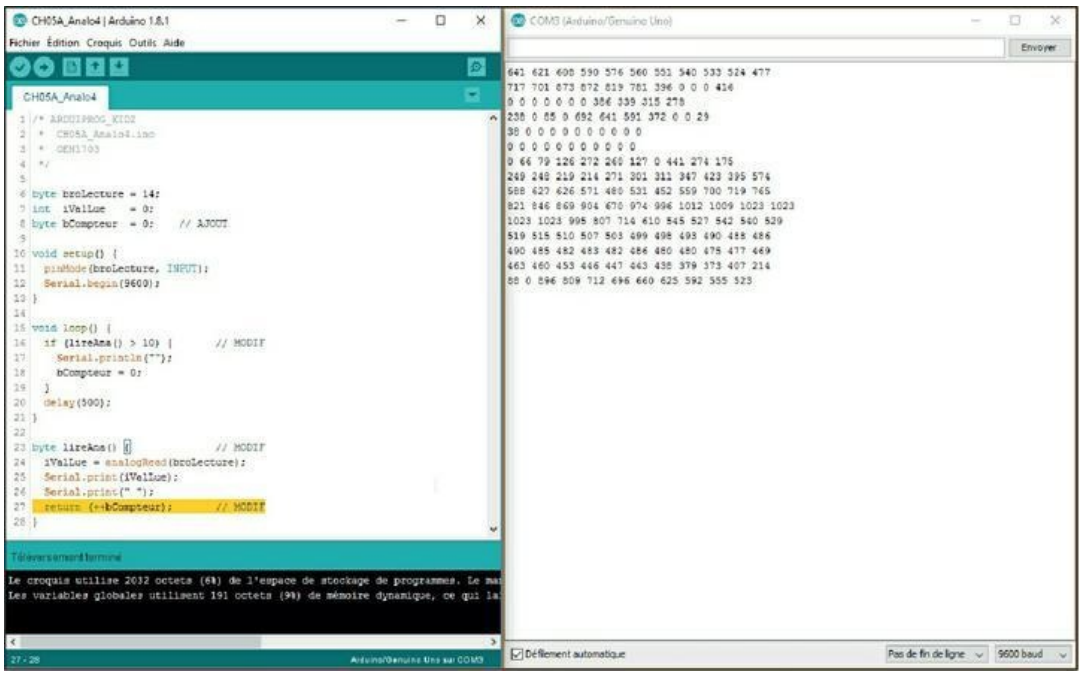

CH05A_Analo4 | Arduino 1.8.1
Fichier Édition Croquis Outils Aide
CH05A_Analo4
/* ARDUIPROG_KIDZ
* CH05A_Analo4.ino
* GEN1703
*/
byte broLecture = 14;
int iValLue = 0;
byte bCompteur = 0; // AJOUT
void setup() {
pinMode(broLecture, INPUT);
Serial.begin(9600);
}
void loop() {
if (lireAna() > 10) { // MODIF
Serial.println("");
bCompteur = 0;
}
delay(500);
}
byte lireAna() { // MODIF
iValLue = analogRead(broLecture);
Serial.print(iValLue);
Serial.print(" ");
return (++bCompteur); // MODIF
}
Le croquis utilise 2032 octets (6%) de l'espace de stockage de programmes. Le ma
Les variables globales utilisent 191 octets (9%) de mémoire dynamique, ce qui la
27 - 28
COM3 (Arduino/Genuino Uno)
Envoyer
641 621 606 590 576 560 551 540 533 524 477
717 701 673 672 619 781 396 0 0 0 426
0 0 0 0 0 0 0 386 339 315 278
238 0 85 0 692 641 591 372 0 0 29
38 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0
0 66 79 126 272 260 127 0 441 274 175
249 248 219 214 271 301 311 347 423 395 574
588 627 626 571 480 531 452 559 700 719 765
821 846 869 904 670 974 996 1012 1009 1023 1023
1023 1023 995 807 714 610 545 527 542 540 529
519 515 510 507 503 499 498 493 490 488 486
490 485 482 483 482 486 480 480 475 477 469
463 460 453 446 447 443 438 379 373 407 214
88 0 896 809 712 696 660 625 592 555 525
Défilement automatique
Pas de fin de ligne
9600 baud

Figure 5.5 : Le projet Analo dans sa dernière version corrigée.

Un thermomètre analogique

Pour ce nouveau projet, je vais d'abord indiquer la liste des composants requis, puis nous procéderons au montage physique du circuit avant de passer à la programmation et aux tests.

Composants requis

Nous n'aurons besoin que d'un seul composant actif dans le projet, un capteur de température de la série LM335 ou LM35. La seule différence entre ces deux modèles est que le LM35 fournit une valeur en degrés Celsius, alors que l'autre fournit une valeur en kelvins. Une simple soustraction permet de passer de l'un à l'autre.

Voici donc les composants requis :

- ta carte Arduino avec son câble USB, évidemment ;
- une plaque d'essai ou d'expérimentation (une de demi-taille suffit) ;
- un capteur de température LM35 ou LM335 ;
- une résistance de 1 kilo-ohm ;
- trois straps de 5 à 10 cm de long, mâle-mâle.

Les composants sont présentés dans la Figure 5.6.

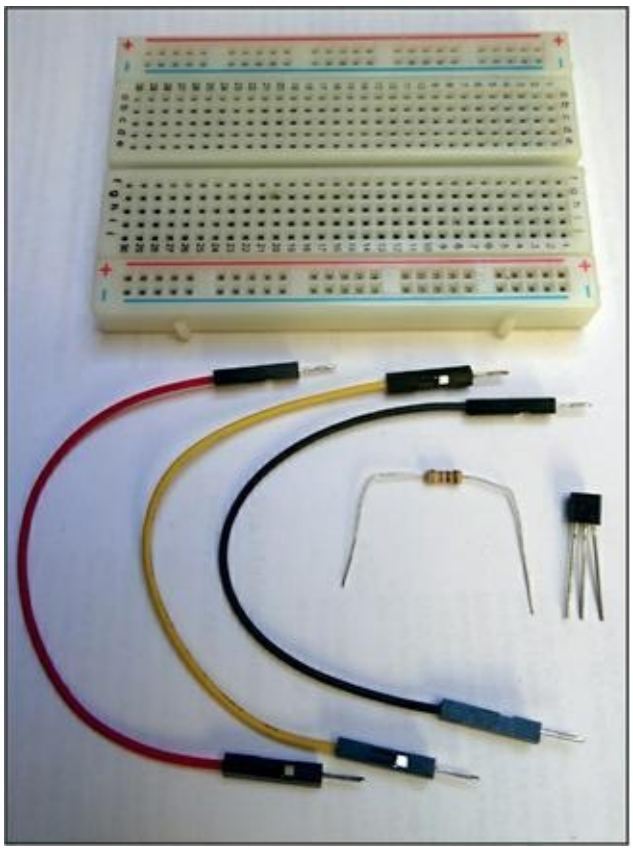


Figure 5.6 : Aperçu des composants du projet Tempera.

La Figure 5.7 montre le schéma de montage sur la plaque d'essai.

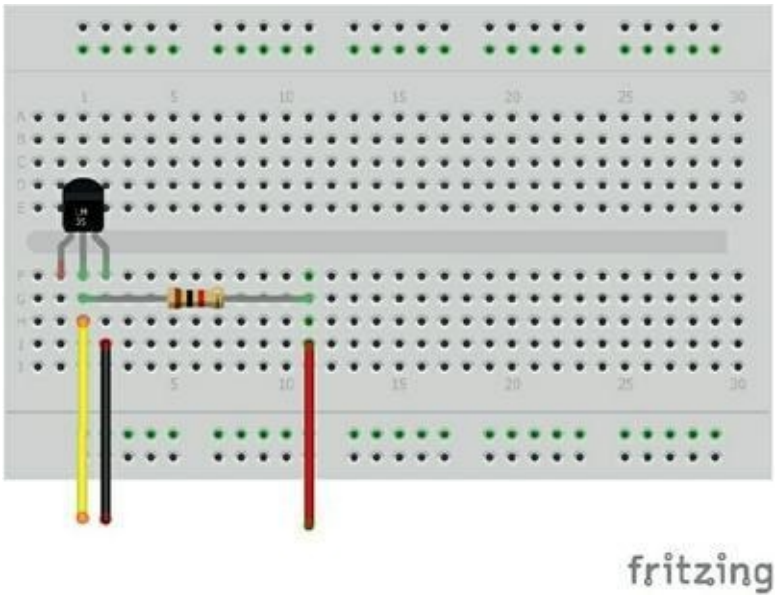


Figure 5.7 : Schéma de montage du projet Tempera.

Quelques remarques, puisque c'est ton premier vrai montage dans ce livre :

- Le capteur doit être positionné en sorte que la face plate sur laquelle est inscrit le nom du modèle soit vers le bas. La patte de gauche ne va pas être utilisée ; elle sert à calibrer la température pour des mesures précises.
- La patte de droite doit être reliée à la masse, c'est-à-dire au 0 V. Sur la carte Arduino, cela correspond aux deux broches GND du connecteur inférieur gauche. Choisis l'une ou l'autre.
- Enfin, la patte centrale du capteur doit être reliée, d'une part, à la broche d'entrée analogique (avec le fil jaune) et, d'autre part, au +5 V, mais en passant d'abord par la résistance de 1 kohm.

Dans mon schéma d'implantation, j'ai fait partir le fil rouge

beaucoup plus à droite, pour avoir assez de place pour insérer les deux pattes de la résistance sans qu'elle soit trop haute au-dessus du circuit, et sans avoir à les raccourcir. En général, il vaut mieux garder les longueurs complètes, car on a souvent besoin de réutiliser les composants.

La Figure 5.8 montre le schéma de principe correspondant.

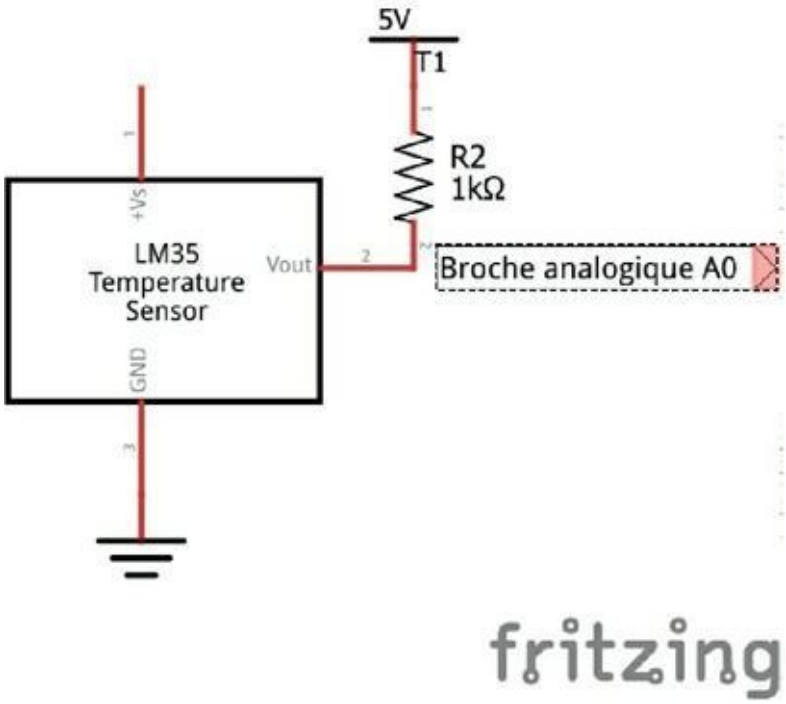


Figure 5.8 : Schéma de principe du projet Tempera.

Et voici le texte source de la première version du projet.

Listing 5.6 : Version 1 du projet Tempera

```
/* ARDUIPROG_KIDZ
 *  CH05B_Tempera1.ino
 *  OEN1703
    Lecture d'un capteur de temperature
LM335 ou LM35
 */
```

```
int broAna = 14;                // Entree
analogique A0
float fMesure = 0.0;
void setup() {
  Serial.begin(9600);
}

void loop() {
  prendreTemp();
  Serial.print("Tension lue en millivolts :
\t");
  Serial.println(fMesure, 0);
  Serial.print("Temperature en Celsius     :
\t");
  Serial.println( (fMesure/10) - 273.15, 1
); // Si LM335 en kelvins
  Serial.println( (fMesure/10), 1 ); // Si
LM35 en Celsius
  Serial.println("-----");
  delay(1000);
}

void prendreTemp() {
  fMesure = (float) analogRead(broAna);
  fMesure = (fMesure*5000) / 1024.0;  //
Conversion de 0-1023 en 0-5V
}
```

Il y a trois nouveautés dans ce listing :

1. Un nouveau type de données, float.

2. L'utilisation du même mot réservé float entre parenthèses, vers la fin du programme.
3. Les deux opérateurs arithmétiques * et / pour la multiplication et la division.

Le type numérique flottant float

Voici la nouvelle déclaration en début de programme :

```
float fMesure = 0.0;
```

Nous avons pour l'instant utilisé trois types numériques, tous entiers :

- le type boolean qui ne permet de stocker que 0 ou 1 ;
- le type int qui permet de stocker une valeur entre 0 et le nombre positif +32767, ou bien un nombre négatif entre –1 et –32768 ;
- le type byte qui permet de stocker une valeur positive entre 0 et 255.

Lorsque tu divises deux nombres et que le résultat n'est pas entier, tu perds de la précision avec des nombres entiers. C'est pourquoi il existe un type permettant de stocker des valeurs non entières. Par exemple, le résultat de 3/2 doit donner 1,5, ou plutôt 1.5 dans nos programmes. C'est une valeur fractionnaire.

Le nouveau type float est donc indispensable, mais il ne faut pas en abuser. Lorsque tu n'as pas besoin de nombres fractionnaires, utilise un type entier. En effet, le type float occupe 4 octets au lieu de 2 pour le type int et 1 octet pour byte. Lorsque tu n'as que deux ou trois variables, cela ne fait pas de différence, mais si tu dois stocker 500 variables, cela occupera presque toute la mémoire

disponible, puisque 500 × 4 = 2000 alors qu'il n'y a que 2048 octets d'espace disponibles.

Lorsque tu veux stocker une valeur dans une variable du type float, ajoute toujours la mention .0 (un point suivi d'un zéro) à la suite de la valeur pour être certain qu'il n'y aura pas de changement de type imprévu, ce que l'on appelle un transtypage.

Observe la ligne suivante dans la boucle principale :

```
Serial.println( (fMesure/10) - 273.15, 1 );
```

Tu remarques que la division par 10 de `fMesure` est entourée par des parenthèses pour que cette opération soit réalisée en premier, avant la soustraction.

Dans les deux premières instructions utilisant `println()`, tu remarques qu'il y a un second paramètre qui vaut soit 0, soit 1. Cela ne concerne que le format de l'affichage et indique le nombre de chiffres à afficher après la virgule. Cela n'a aucun effet sur la valeur de la variable

Tu remarques enfin dans notre fonction `prendreTemp()`, tout en bas, que nous prenons la même précaution : nous délimitons l'opération à réaliser en premier par un jeu de parenthèses.

Saisie et test du projet

Je te propose maintenant de saisir ce texte source puis de le tester.

Comme pour notre essai de température interne du mikon, dès que quelques mesures sont affichées, approche délicatement un doigt du capteur et touche-le pour transmettre un peu de ta chaleur. Tu dois voir la valeur croître. Elle doit redescendre lorsque tu éloignes le doigt.

Puisque je te parle souvent d'occupation mémoire, intéressons-nous à un style de données qui prend beaucoup de place : les textes des messages.

À la chasse au gaspi !

Le projet comprend deux messages fixes :

```
  Serial.print("Tension lue en millivolts :
\t");
…
  Serial.print("Temperature en Celsius     :
\t");
```

Voyons combien chacune consomme d'espace mémoire.

1. Dans le texte source, positionne-toi après le premier guillemet et donc avant la lettre T de `Tension` dans le premier message.
2. Maintiens la touche Majuscule enfoncée tout en utilisant la flèche du curseur Droite et en comptant le nombre de caractères. Si tu as saisi le message exactement comme moi, tu dois compter 30 caractères. Ces caractères sont enchaînés les uns à la suite des autres, et c'est pourquoi on appelle cela une *chaîne de caractères* (*string* en anglais).

\t est un métacaractère

Les deux messages se terminent par un couple de caractères étrange : une barre oblique inverse (une antibarre) et la lettre *t*. C'est tout simplement le symbole d'une commande pour avancer vers la marge droite de quelques positions (un saut de colonne). Cela permet de bien aligner les valeurs numériques.

Une fois que tu as compilé le projet, les deux lignes du panneau inférieur indiquent par exemple que le croquis utilise :

- 3644 octets de la mémoire du programme ;
- 266 octets de la mémoire des données.

Une astuce permet de demander de stocker les chaînes de caractères dans la mémoire du programme, puisque les messages ne peuvent pas changer au cours de l'exécution (ce sont des chaînes littérales). Ce restockage est impossible pour les « vraies » variables.

La notation F()

Pour qu'une chaîne constante soit stockée dans la mémoire flash réservée au code exécutable, il faut entourer la chaîne d'un couple de parenthèses précédé de la lettre F en majuscule.

Voici comment tester cette technique d'économie de la mémoire :

1. Relance si nécessaire une compilation sans avoir rien changé au texte source afin de pouvoir noter l'occupation mémoire du programme et des données. Dans mon exemple, c'est 3646 pour le programme et 268 pour les données.
2. Place-toi avant le guillemet ouvrant de la première chaîne et insère une espace, la lettre F majuscule et une parenthèse ouvrante. Voici le résultat :

   ```
   Serial.print( F("Tension lue en millivolts
   : \t") ;
   ```

3. Place-toi entre le second guillemet et la parenthèse de la même ligne, et insère une autre parenthèse, suivie d'une espace pour améliorer la lisibilité.

4. Fais de même pour l'autre chaîne, `Temperature en Celsius…`

5. Éventuellement, avec la commande Fichier/Enregistrer sous, implante le texte dans un nouveau fichier de projet.

Voici à quoi doit ressembler le programme. Les deux lignes modifiées sont imprimées en italique :

Listing 5.7 : Version 2 de Tempera

```
/* ARDUIPROG_KIDZ
 *  CH05B_Tempera2.ino
 *  OEN1703
    Lecture d'un capteur de temperature
LM335 ou LM35
 */

int broAna = 14;  // Entree analogique A0
float fMesure = 0.0;

void setup() {
  Serial.begin(9600);
}

void loop() {
  prendreTemp();
  Serial.print( F("Tension lue en millivolts
: \t") );
  Serial.println(fMesure, 0);
  Serial.print( F("Temperature en Celsius
```

```
: \t") );
  Serial.println( (fMesure/10) - 273.15, 1
); // Si LM335 en kelvins
  Serial.println( (fMesure/10), 1 );
// Si LM35 en Celsius
  Serial.println("-----");
  delay(1000);
}

void prendreTemp() {
  fMesure = (float) analogRead(broAna);
  fMesure = (fMesure*5000) / 1024.0;  //
Reconversion 0-1023 en 0-5V
}
```

Relance la compilation et compare l'occupation mémoire avec la précédente. Normalement, nous n'occupons plus 268, mais 208 octets du précieux espace limité disponible pour les variables. C'est tout à fait normal, puisque nous avons réussi à stocker ailleurs deux chaînes de caractères de 30 octets chacune.

En réalité, les 30 octets ne sont pas ceux que tu crois. En effet, les deux caractères (donc 2 octets) \t forment un symbole qui est transformé en un seul octet ; c'est un code de contrôle. Mais un autre octet est ajouté automatiquement à la fin de chaque chaîne : c'est la valeur 0. Ce 0 terminal sert en langage C à savoir où se termine une série de caractères constituée sous la forme d'une chaîne. On appelle cela une chaîne AZT.

À part cette optimisation de l'occupation mémoire, le programme doit fonctionner exactement de la même manière.

Nous en avons terminé avec ce projet de capture de température. Dans le prochain chapitre, nous allons découvrir comment contrôler le monde extérieur grâce aux sorties numériques.

Récapitulons

Dans ce chapitre, nous avons appris :

- à exploiter une entrée analogique ;
- ce qu'est un convertisseur analogique-numérique ;
- à déclarer une variable numérique ;
- les types de données byte, boolean, int et float ;
- à créer une fonction et à exploiter des fonctions prédéfinies ;
- à faire renvoyer une valeur d'une fonction avec return ;
- à optimiser l'espace mémoire avec la notation F( ).

Chapitre 6
Parler, c'est agir

AU MENU DE CE CHAPITRE :

- **La LED fait de la résistance**
- **Les broches de sortie numériques**
- **Les lucioles dansent**
- **Le hasard qui fait des choses**

Les deux précédents chapitres nous ont permis de voir comment le mikon s'organise pour être à l'écoute du monde extérieur, en utilisant ses broches comme des yeux et des oreilles, le mode entrée, numérique puis analogique. Voyons maintenant comment le mikon peut « parler » au monde.

Dans le mot « microcontrôleur » , tu sais maintenant à quoi correspond la partie « micro » : il s'agit de cette petite lamelle de silicium de 3 mm de côté sur quelques feuilles de papier d'épaisseur. L'autre partie, « contrôleur » , indique qu'il s'agit de contrôler, donc d'influer sur le cours de certains événements extérieurs à la puce.

Lorsque tu entends quelque chose qui t'intéresse, cela te fait penser à d'autres choses, et parfois te pousse à déclencher une action.

Quand tu réalises une action, tu contrôles quelque chose, avec tes mains souvent, avec tes pieds si tu jongles avec un ballon de football. Pour donner un ordre ou un avis, tu utilises ta bouche.

Le mikon a de tout petits bras. Il travaille avec de l'énergie électrique, mais la quantité d'énergie qu'il peut fournir est très

limitée. Il est naturellement conçu pour donner des ordres à des assistants que l'on appelle des actionneurs.

Même un président de la République, l'homme le plus puissant de son pays, ne peut pas soulever une pierre rien qu'en parlant. En revanche, il peut donner l'ordre de déclarer la guerre à un pays qui le menace.

Pour contrôler l'environnement avec le mikon, tu vas le faire parler (généralement en silence) sur une ou plusieurs de ses broches utilisées comme sorties ; ces broches vont passer de 0 à 1, et rien d'autre. Le mikon offre 14 broches de sortie numériques, identifiées de 0 à 13 (celles marquées DIGITAL).

Pour commencer notre découverte des moyens de contrôle de l'environnement du mikon, je te propose un cas particulier : se servir d'une broche de sortie du mikon pour non seulement transmettre des ordres, mais directement réaliser l'action, c'est-à-dire alimenter en énergie l'objet que nous voulons contrôler.

Pour une autre distinction entre ordre et action, imagine ceci : tu construis un château de cartes sur une table, puis tu t'en approches et tu parles assez fort pour le faire s'écrouler uniquement par la parole. Mais tu imagines bien qu'il y a des limites à la puissance de destruction de la parole (souffler, c'est tricher).

Dans le monde entier, les sessions de formation Arduino commencent par le projet consistant à allumer et éteindre une diode LED. Dans cet exemple universel, la diode reçoit son énergie du mikon, alors que le mikon devrait se contenter de transmettre des ordres à un interrupteur électronique qui mettrait la diode sous tension et hors tension.

Pour me plier à la coutume, je vais commencer par cet exemple universel et contrôler une diode LED depuis une broche de sortie Arduino.

Une belle diode LED

Puisque, dans ce chapitre, j'ai décidé de faire la lumière sur plusieurs sujets, intéressons-nous au composant qui va nous éclairer : la diode électroluminescente, que l'on appelle plutôt diode LED.

C'est d'abord une diode, c'est-à-dire un composant constitué de deux tranches de silicium. L'une des tranches est un peu positive (elle manque d'électrons), l'autre plutôt négative (elle a un peu trop d'électrons). Les deux tranches sont collées l'une contre l'autre, ce qui donne une jonction. Le résultat est que le courant ne peut traverser la jonction que dans un sens. Lorsque tu la branches à l'envers, il ne se passe rien, comme si le fil était coupé à l'intérieur. Quand tu la branches dans le bon sens, elle va laisser passer le courant. Et puisque c'est une diode spéciale, une diode LED, elle va émettre de la lumière. La seule condition est que la tension soit suffisante.

La tension de seuil d'une LED dépend de la couleur qu'elle émet :

- Pour une LED rouge, la tension de seuil est d'environ 1,8 V. Pour une LED verte ou jaune, elle est d'environ 2 V.
- Pour une LED bleue, la tension minimale qui lui permet de s'allumer est d'environ 3 V.
- Enfin, pour une LED blanche, il faut fournir environ 3,5 V.

Si une LED émet la lumière, elle consomme du courant. Elle transforme de l'énergie électrique en énergie lumineuse et en chaleur. Les LED ne sont pas gourmandes : pour voir briller une LED en plein jour, il suffit de lui faire traverser un courant de 10 à 20 milliampères (abrégés en mA). Le problème, c'est qu'elle ne sait pas se retenir.

Comparons avec une lampe à incandescence, ce genre de lampe qui était universelle jusqu'à la fin du dernier millénaire. Supposons une lampe dont la puissance est égale à 100 watts. Si elle est alimentée

avec l'électricité de la maison (arrondissons à 200 volts), elle consommera 500 mA. La formule est assez simple :

```
Puissance = tension X courant, soit P = U.I
```

Une fois qu'elle consomme ses 500 milliampères, la lampe à incandescence est contente : son filament rougit et elle éclaire la pièce. Elle se stabilise à cette puissance. Une LED ne fonctionne pas selon le même principe : elle ne sait pas facilement s'arrêter de consommer du courant. Voilà pourquoi il faut freiner ses ardeurs en plaçant toujours un frein à électrons dans le même circuit que la LED. C'est une résistance.

Avant de découvrir comment choisir la bonne résistance, encore un petit détail concernant les LED : puisque l'on risque de les brancher à l'envers, il faut pouvoir distinguer les deux côtés. Cela s'appelle un détrompage. Pour une LED, un côté porte le nom « cathode » et l'autre, le nom « anode » . La cathode doit toujours être branchée du côté du 0 V (la masse, GND). Bien sûr, l'anode est à brancher du côté du pôle positif, soit le +5 V ou une autre tension positive.

Lorsque tu prends la LED entre les doigts, la patte la plus courte est la cathode. Pour t'en souvenir, dis-toi que court et cathode commencent par la même lettre.

Figure 6.1 : Une LED avec ses deux pattes de longueurs différentes.

Pour une LED que tu veux réutiliser, si les deux pattes ont été coupées à la même longueur, il te reste une solution : le boîtier de la LED est aplati du côté de la cathode.

Sur les schémas de principe, le symbole de la LED est le suivant :

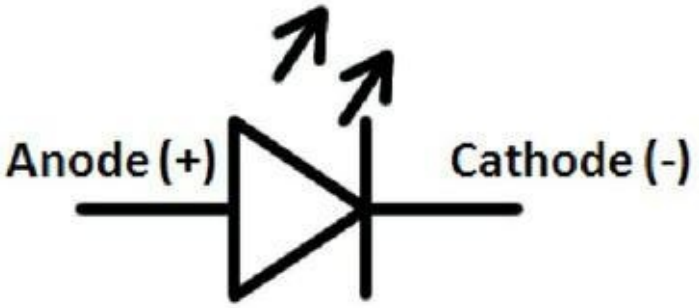


Figure 6.2 : Schéma du composant LED.

C’est un triangle avec un trait en travers du sens du courant. Lorsque tu regardes le symbole à l’horizontale, tu peux deviner un K majuscule, deux côtés du triangle formant les deux traits en biais de la lettre K. Du côté gauche de ce K, c’est la cathode.

Puisqu’il nous faut protéger la LED, ainsi que la broche de sortie du mikon, voyons comment utiliser une résistance.

Savoir faire tomber la tension

Limitons les dégâts

Chacune des broches numériques supporte au grand maximum de fournir un courant de 40 milliampères. Si le composant que tu branches sur une de ces broches consomme plus de courant, tu risques de détruire la broche, le mikon, voire la totalité de la carte Arduino.

Les broches d’entrée analogiques A0 à A5 peuvent en fait aussi être utilisées comme broches de sortie numériques sous les numéros 14 à 19.

Pour ne pas risquer de détruire notre cher mikon, je te propose un petit cours d’électronique. N’aie aucune crainte, je vais rester très clair.

Dans une rivière, l’eau va dans le sens de la descente à cause de l’attraction terrestre. En électricité, l’endroit le plus haut correspond

au pôle positif et l'autre, au pôle négatif. Si tu laisses faire les électrons, ils vont aller le plus vite possible de l'endroit où ils sont trop nombreux vers l'endroit où ils manquent. Pour de l'eau, c'est l'équivalent d'une cascade : elle détruit tout sur son passage. En électricité, c'est un court-circuit, qui fait aussi des dégâts.

Il s'agit donc de contrôler la pente, c'est-à-dire la vitesse du courant. C'est la tension, et elle est mesurée en volts (V). Le composant magique qui va nous permettre de réduire la tension, et donc la quantité de courant, s'appelle une résistance.

La résistance est le composant électronique le plus simple et le meilleur marché qui soit. Elle n'a qu'un seul rôle : transformer de l'énergie électrique en chaleur. La résistance décide de la quantité de courant qui passe du pôle positif au pôle négatif d'un circuit. Si tu osais connecter le pôle + et le pôle – d'une pile avec un bout de fil (ne fais jamais cela !), toute l'énergie en trop d'un côté foncerait vers le côté négatif, le plus vite possible. Le fil rougirait puis fondrait, et la pile pourrait même exploser.

En revanche, si tu réunis les deux bornes d'une pile de 1,5 V avec une résistance de 100 ohms, par exemple, la pile va se vider lentement. Cela grâce à la formule magique du chercheur allemand dont Ohm était le nom de famille. Voici cette formule :

```
tension = résistance X courant
```

```
U = R . I
```

En la transformant, on peut trouver la valeur de la résistance :

```
R = U/I
```

Autrement dit, s'il y a 5 V sur une broche numérique et que l'on ne veuille consommer au maximum que 20 milliampères pour ne pas endommager le mikon, il faut faire le calcul suivant :

```
5V divisés par 0,02 A = 250 ohms
```

Si nous faisions ce montage, une des pattes de la résistance serait branchée sur la sortie numérique et on verrait une tension de 5 V. L'autre patte de la résistance, branchée sur la broche de masse GND, serait à la tension nulle de 0 V. Autrement dit, la résistance fait chuter la tension de 5 V. Cela s'appelle une différence de potentiel.

Il y a une dernière chose à vérifier quand on choisit une résistance : si elle ne va pas trop chauffer. Pour calculer la puissance qui va être consommée par une résistance, il suffit d'appliquer une autre formule magique très simple qui découle de la précédente :

```
Puissance = tension X courant
P = U . I
```

Dans notre exemple, la tension est de 5 V et le courant est de 0,02 A. La puissance que va dégager la résistance sous forme de chaleur est donc de :

```
P = 5*0,02 = 0,1 soit 100 milliwatts
```

Les résistances que l'on utilise en électronique supportent une puissance maximale de 500 mW ou de 250 mW. On parle également de demi-watt et de quart de watt.

Il ne reste plus qu'à identifier la bonne résistance une fois que tu as fait le calcul. Si tu as acheté les composants permettant de réaliser les projets de ce livre, tu as sans doute déjà remarqué ces petits objets qui ressemblent à des insectes et portent des anneaux colorés. Ce sont les résistances. Les anneaux de couleur servent à en indiquer la valeur. Voyons comment lire ces anneaux.

Haussez les couleurs !

Les chercheurs qui ont défini le code de couleurs des résistances se sont inspirés de l'arc-en-ciel. Chaque couleur correspond à un chiffre. Là où les choses se corsent, c'est que le nombre d'anneaux

n’est pas toujours le même. C’est une question de qualité de fabrication.

Si tu achètes des résistances dans un magasin ou sur un site Web, on te livrera des résistances avec une tolérance de précision de 5 %. Cela signifie que si tu demandes une résistance de 100 ohms, celle que tu vas obtenir pourra faire 95 ou 105 ohms. Cela n’a aucune importance, car les valeurs n’ont souvent pas besoin d’être absolument exactes. Le projet fonctionnera de la même manière.

Pour marquer une résistance avec une tolérance de 5 %, on ajoute à la résistance un anneau de couleur or un peu à l’écart des autres. Avec cette tolérance, il suffit de trois autres anneaux pour coder la valeur :

- Le premier anneau correspond au premier chiffre.
- Le deuxième anneau correspond au deuxième chiffre.
- Le troisième anneau indique le nombre de zéros après les deux premiers chiffres.

Voici tout d’abord comment décrypter les chiffres :

0	noir
1	marron
2	rouge
3	orange
4	jaune
5	vert
6	bleu
7	violet
8	gris

9	blanc

Passons à la pratique : tu repères d'abord l'anneau argenté ou doré et tu le places du côté droit devant toi. Tu lis ensuite les trois anneaux de couleur en partant du côté gauche. Prenons l'exemple de la Figure 6.3.

Figure 6.3 : Exemple de résistance.

Dans cet exemple, les couleurs sont rouge, rouge, marron. Soit 2, puis 2, puis 1. C'est la valeur 22 suivie d'un seul zéro, donc 220 ohms. S'il y avait du noir à la place du marron, il aurait zéro 0, la valeur serait donc de 22 ohms.

Tu rencontreras rarement d'aussi faibles valeurs dans tes projets. En général, les résistances que l'on utilise dans le monde Arduino ont des valeurs entre 100 ohms et 1 million d'ohms, c'est-à-dire 1 Mohm (mégaohm).

Pour te souvenir de l'ordre des couleurs, tu peux mémoriser la phrase suivante :

```
Ne Mangez Rien Ou Je Vous Brûle Votre Grande Barbe
```

```
Noir Marron Rouge Orange Jaune Vert Bleu
Violet Gris Blanc
```

La première lettre de chaque mot permet de se souvenir de l'ordre croissant des couleurs.

Une résistance comprenant cinq ou six anneaux est une résistance de très grande précision. Dans ce cas, il y a quatre ou cinq anneaux pour les chiffres et un anneau pour la tolérance. Nous n'aurons pas besoin d'une telle précision dans les projets.

Une dernière chose concernant les résistances, avant de passer au montage. Tu ne peux pas obtenir n'importe quelle valeur auprès des fabricants de résistances. Des séries ont été normalisées en fonction de la précision. Voici les valeurs que l'on peut acheter dans la série qui nous intéresse, la série E24 de la tolérance 5 %. Je prends comme exemple la plage de valeurs entre 100 et 1000 ohms :

```
100, 110, 120, 130, 150, 160, 180, 200, 220,
240, 270, 300,

330, 360, 390, 430, 470, 510, 560, 620, 680,
750, 820, 910 et 1000
```

Autrement dit, si tu fais un petit calcul et si la résistance dont tu as besoin est de 250 ohms, tu ne trouveras pas ce composant. Il n'existe qu'en 240 ou en 270 ohms. Par mesure de précaution, tu prendras toujours une valeur légèrement supérieure, donc ici 270 ohms.

Passons maintenant aux exercices pratiques.

Un projet de luciole

Commençons par le montage du projet. Voici les composants qu'il te faut :

- la carte Arduino avec son câble USB ;
- une diode LED rouge, verte ou bleue ;

- une résistance de 220 ou 270 ohms ;
- un strap noir mâle-mâle ;
- un strap rouge mâle-mâle ;
- une petite plaque d'expérimentation : une de demi-taille (400 trous) suffira.

Le montage

1. Prends ta diode LED et insère la patte la plus courte, celle de la cathode, dans la rangée d'alimentation du 0 V marquée d'un **–** (souvent munie d'un filet bleu sur toute la longueur) sur le bord de la plaque, puis écarte un peu les deux pattes (sans forcer) pour pouvoir insérer l'autre patte, celle de l'anode, dans le premier trou à l'intersection du **a** et du **1** sur la plaque. Aide-toi de la Figure 6.4.

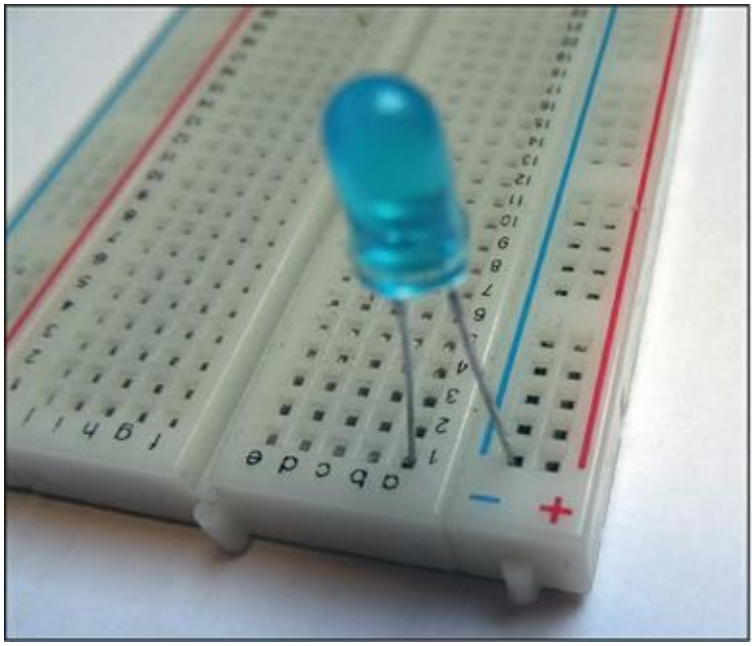


Figure 6.4 : La LED est insérée.

2. Prends ta résistance et insère une de ses pattes dans le trou à côté de l'anode, dans la même rangée de cinq trous. Insère l'autre patte à la même hauteur dans n'importe quel trou de l'autre demi-rangée, sauf dans le plus éloigné.

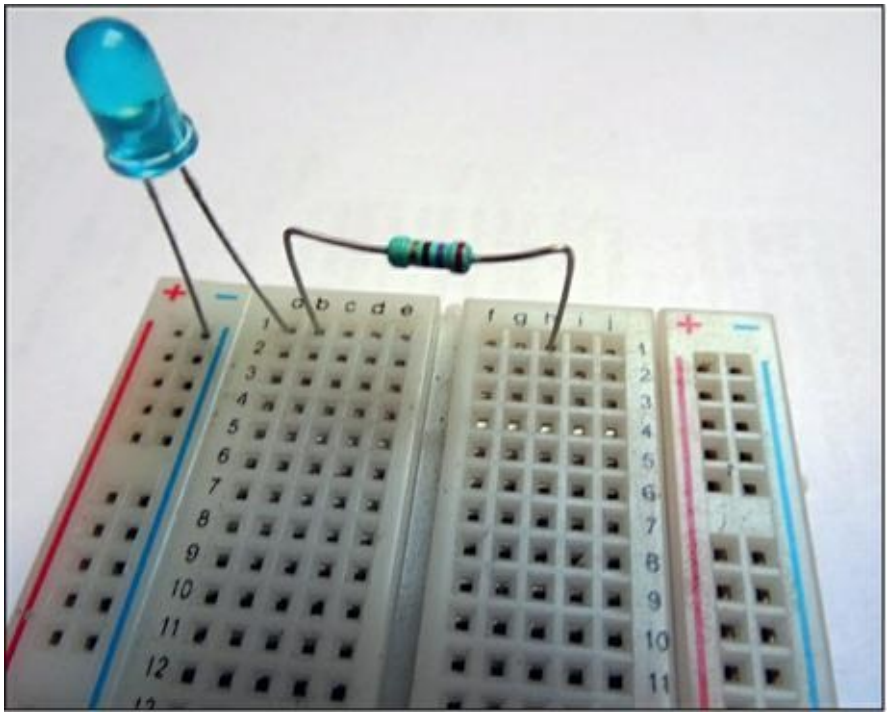


Figure 6.5 : La résistance est en place.

3. Prends le strap noir et insère-le dans la ligne du 0 V, celle avec le signe -, dans le trou suivant, celui où tu viens d'insérer la cathode de la diode LED. Insère l'autre extrémité du strap (la carte Arduino n'est pas sous tension !) dans une des deux broches marquées GND sur le connecteur du bas de la carte Arduino.

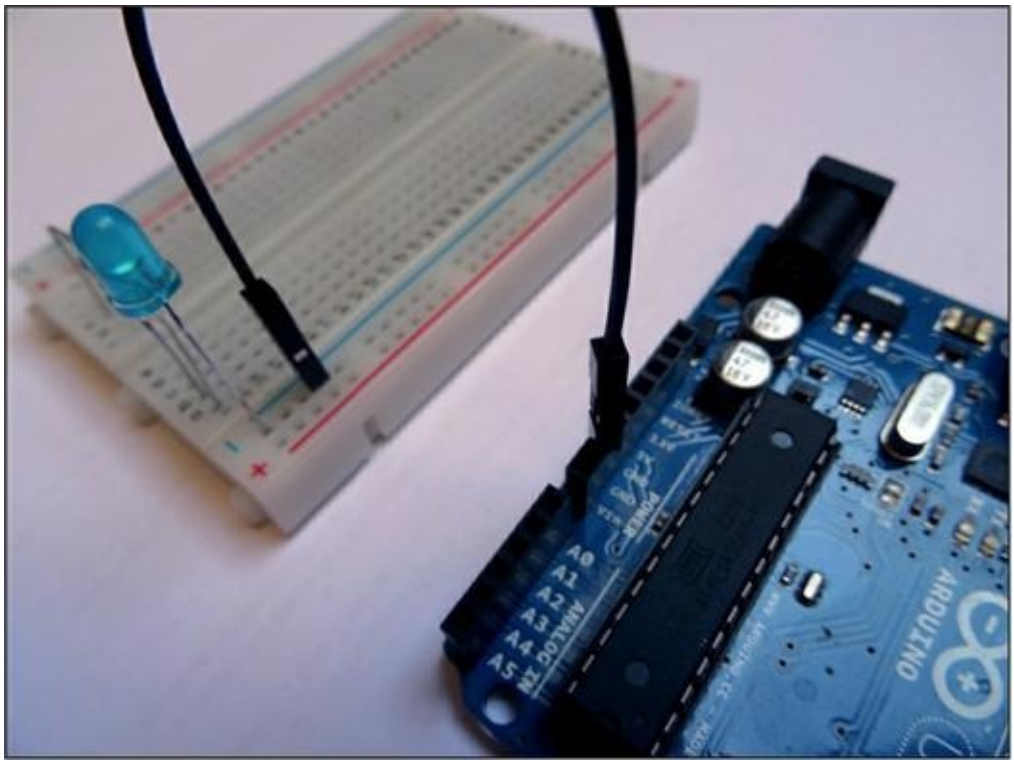


Figure 6.6 : La cathode est reliée à la masse GND.

4. Prends le strap rouge et insère une des extrémités à côté de la résistance, par exemple dans le trou J que je t'ai justement invité à laisser libre.

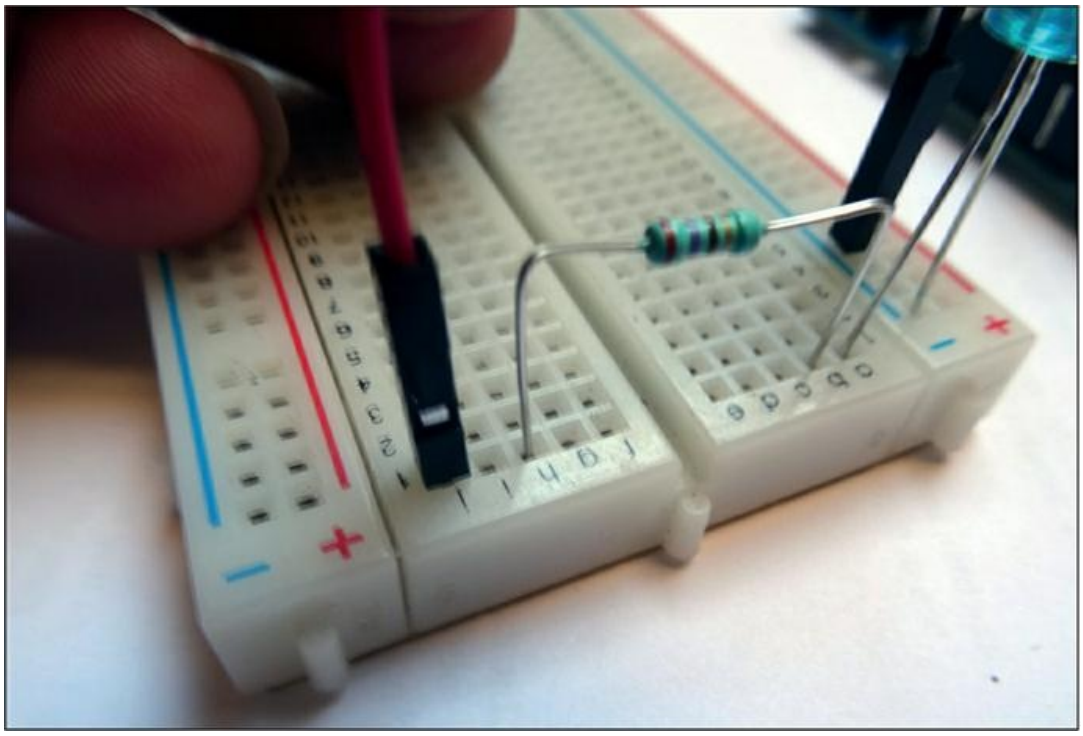


Figure 6.7 : Branchement du fil d'alimentation +5 V sur la plaque.

5. Il ne reste qu'à insérer l'autre extrémité du strap rouge dans la broche DIGITAL 11 de la carte Arduino.

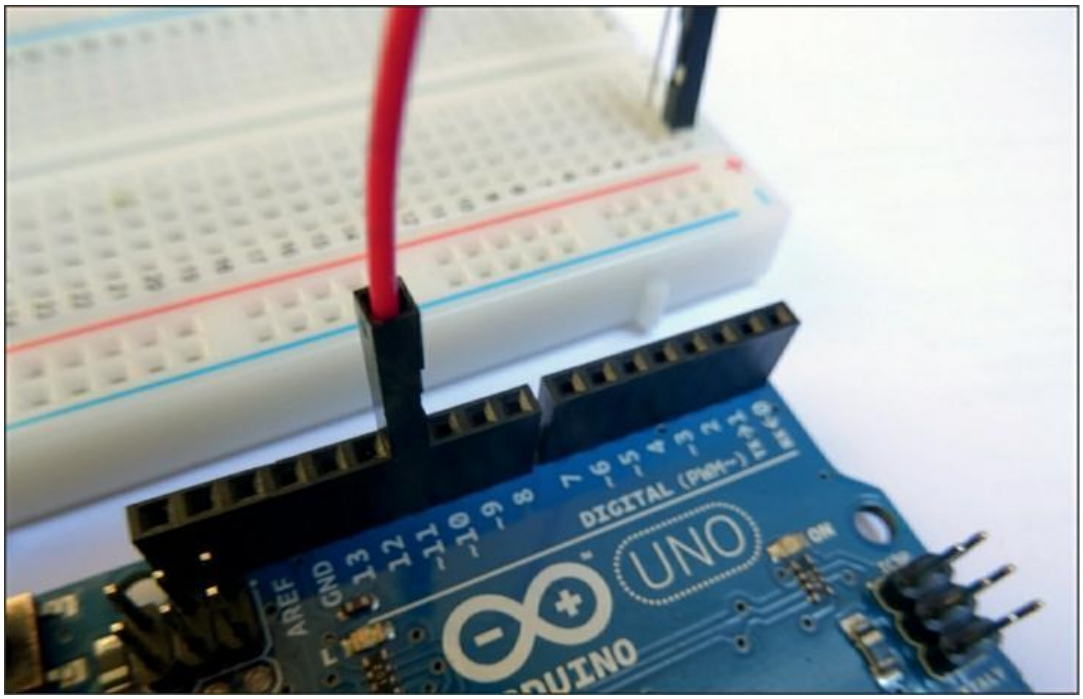


Figure 6.8 : Dernier branchement au +5 V.

Le montage est maintenant terminé. Prends le temps de vérifier toutes les connexions.

La Figure 6.9 montre le plan de montage correspondant.

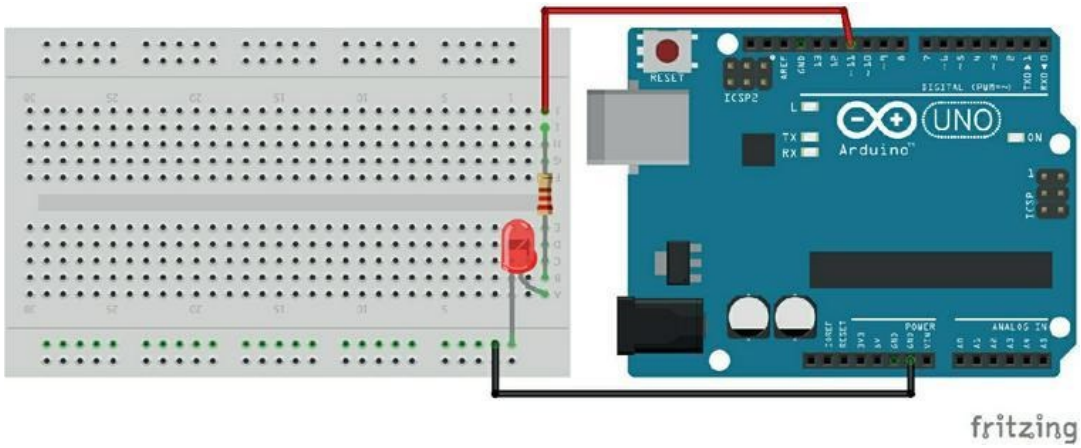


Figure 6.9 : Plan de montage du projet.

Et la Figure 6.10 illustre le principe correspondant à ce projet :

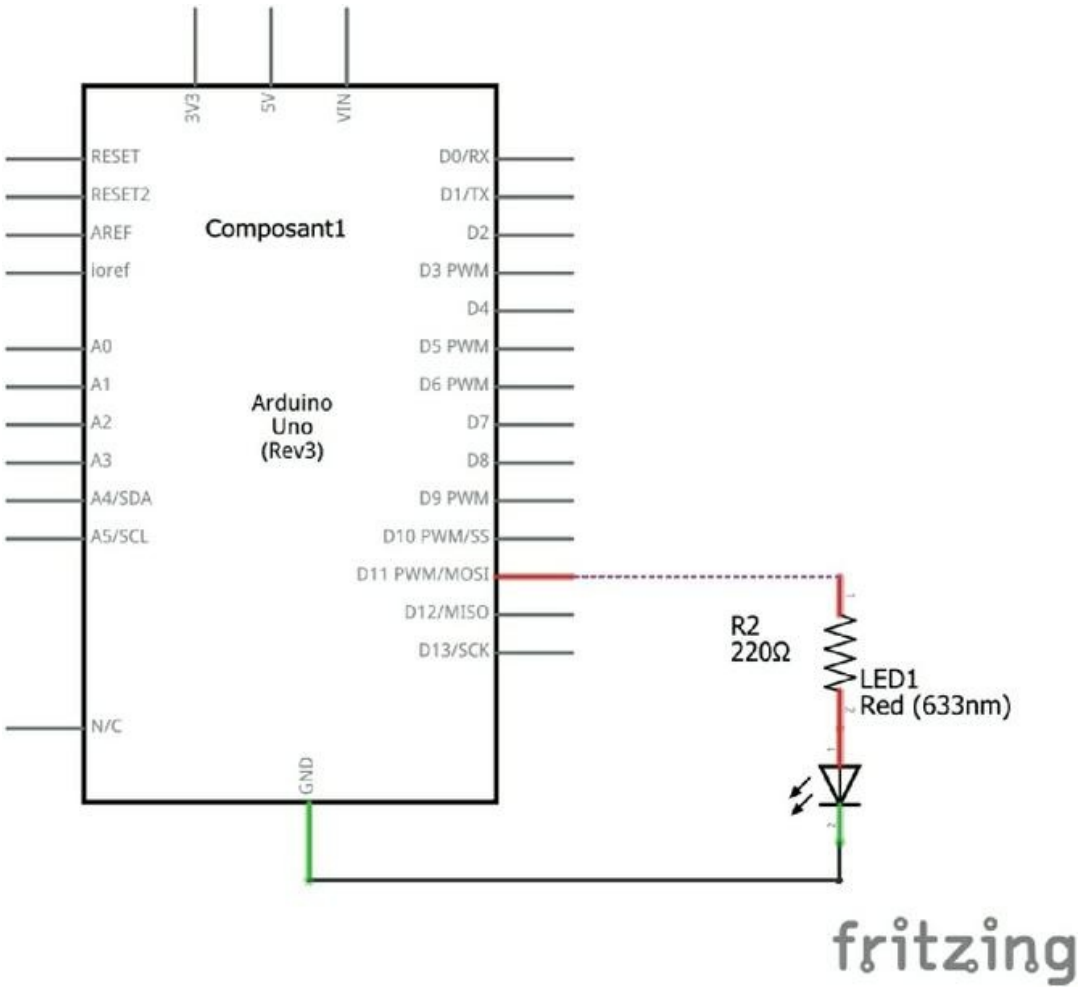


Figure 6.10 : Schéma de principe du projet.

Voilà bien longtemps que nous n'avons pas programmé. Alors allons-y !

Codons notre luciole

Une fois n'est pas coutume, nous allons partir d'un programme existant. Il ne nous fera pas gagner beaucoup de temps, mais cela te montrera le principe : n'hésite pas à entrer en communication avec la communauté des passionnés d'Arduino du monde entier. Tu pourras partager tes projets et profiter de ceux des autres.

Des souris et des ohms

Tu peux te demander comment je suis arrivé à 250 ohms pour la résistance. Voici les détails du calcul.

Je suppose que nous utilisons une LED rouge dont la tension minimale (le seuil) est de 2 V. Si on dispose d'une alimentation de 5 V comme celle des broches numériques du mikon, il faut laisser 2 V à la diode. La résistance doit donc faire chuter la tension de 3 V. Si je veux limiter le courant à 20 mA, il faut appliquer la formule suivante :

```
R = 3 V

    ----

    0,02

soit 150 ohms
```

En choisissant environ 250 ohms, je donne une bonne marge de sécurité. La diode brillera malgré tout, mais un peu moins fort.

1. Démarre l'atelier Arduino si nécessaire.
2. Ouvre le menu **Fichier** et choisis le sous-menu **Exemples** puis le menu **1.Basics**. Sélectionne enfin dans ce sous-menu le troisième exemple, `Blink`.

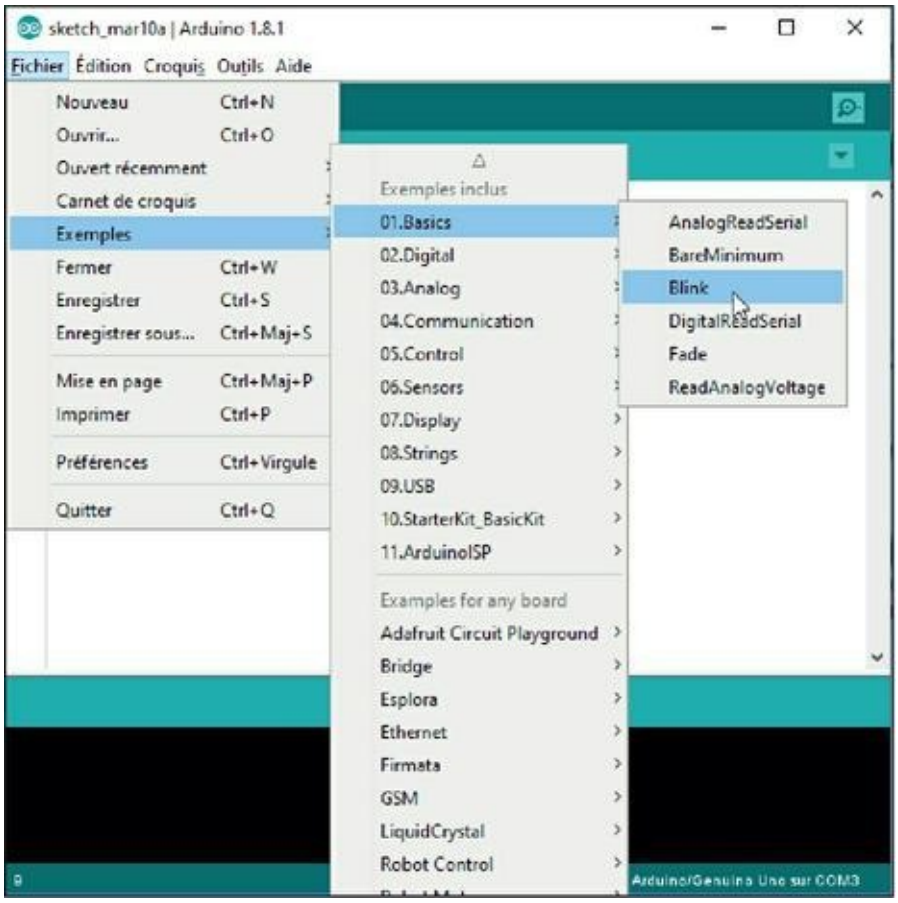


Figure 6.11 : Réutilisation d'un exemple prédéfini.

3. Nous allons commencer par un petit nettoyage. Tu constates qu'il y a un certain nombre de lignes de commentaires en anglais. Nous allons toutes les supprimer. Place-toi au début de la deuxième ligne et sélectionne avec la souris jusqu'à celle qui précède les deux caractères */ qui marquent la fin du bloc de commentaires.

   Prends la bonne habitude d'ajouter un commentaire pour ton nouveau projet, par exemple le nom du projet, `Luciole`.

4. Procède immédiatement à la création d'un nouveau dossier pour ce projet au moyen de la commande **Fichier/Enregistrer sous**. Choisis un nom cohérent avec ceux que tu as déjà utilisés pour les projets précédents. Le fichier d'exemple correspondant porte le nom suivant :

   `CH06A_Luciole1`

5. Supprime les autres lignes de commentaires qui se trouvent au début et dans les deux fonctions obligatoires. Ton projet devrait avoir à peu près le même aspect que dans la Figure 6.12.

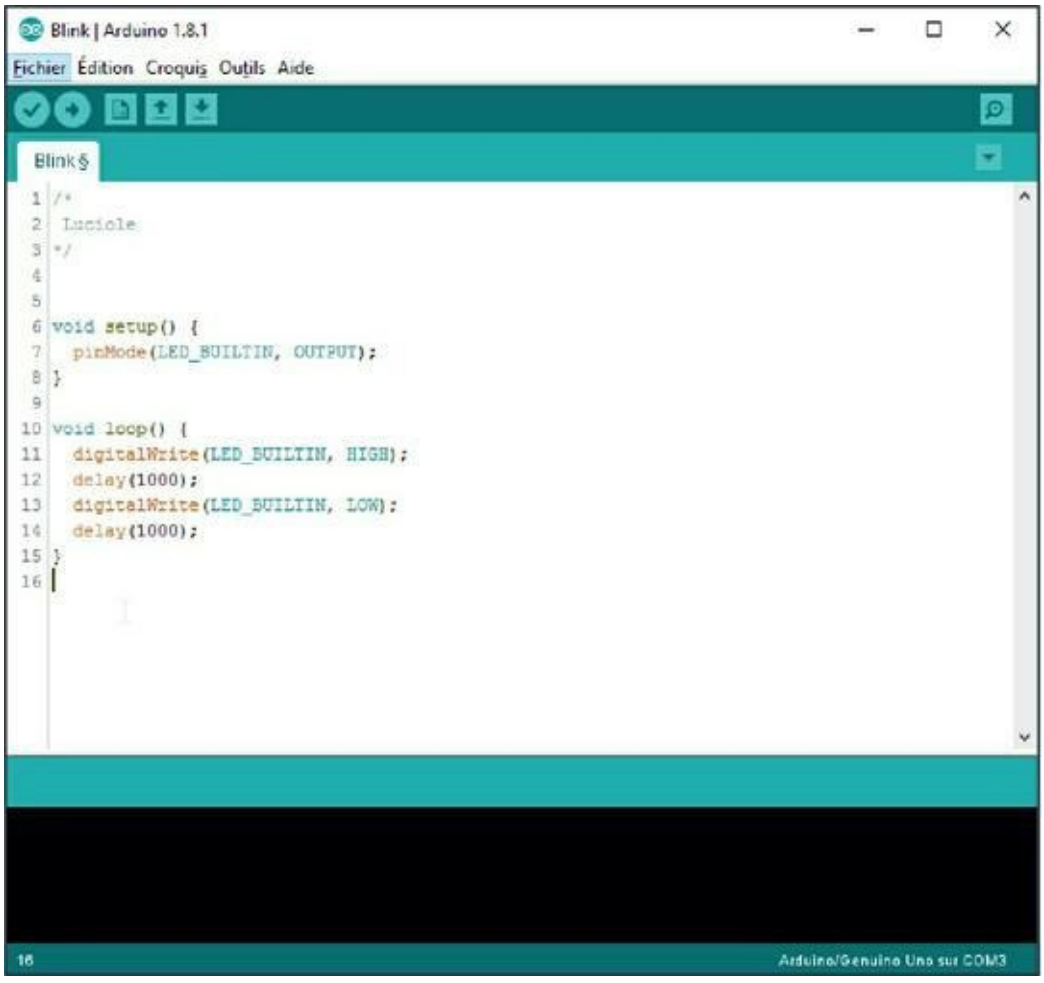


Figure 6.12 : Le projet Luciole dans sa version initiale.

6. Trois appels de fonctions utilisent le mot LED_BUILTIN. Il s'agit d'une constante prédéfinie dans l'atelier. Elle symbolise tout simplement le numéro de la broche 13, celle sur laquelle est connectée la petite LED orange sur la carte. Mais nous avons choisi de brancher notre LED externe sur la broche 11.

   Sélectionne tour à tour les trois exemplaires du mot LED_BUILTIN, par exemple en double-cliquant dans le mot, et saisis à la place la valeur numérique 11, celle de la broche que nous utilisons pour contrôler notre LED.

Le projet doit avoir le même aspect que le Listing 6.1.

Listing 6.1 : Le projet Luciole1

```
/* ARDUIPROG_KIDZ
 *  CH06A_Luciole1
 *  OEN1710
 */

void setup() {
  pinMode(11, OUTPUT);
}

void loop() {
  digitalWrite(11, HIGH);
  delay(1000);
  digitalWrite(11, LOW);
```

```
  delay(1000);
}
```

Tu peux maintenant utiliser le bouton ou la commande **Téléverser** pour compiler et téléverser le programme vers le mikon.

Comme prévu, tu vois clignoter la LED à raison d'un changement par seconde. Il n'y a rien de bien nouveau pour l'instant : nous utilisons la fonction prédéfinie `digitalWrite()` qui réclame deux paramètres d'entrée : le numéro de la broche et la valeur à y écrire. Les deux mots réservés HIGH et LOW sont des équivalents des valeurs 1 et 0.

Voyons maintenant comment ajouter une deuxième LED en faisant alterner l'allumage entre les deux LED.

Deux lucioles qui dansent (Luciole2)

Dans cette deuxième version, nous allons travailler de façon plus confortable. Au lieu d'indiquer directement le numéro des broches, nous allons déclarer des variables en début de programme.

1. Enregistre si tu le désires le projet sous un nouveau nom avant de le modifier.
2. Place-toi juste après le commentaire initial, donc avant la ligne de tête de la première fonction `setup()`, puis insère une ligne d'espace avec la touche Entrée.
3. Insère la ligne de déclaration de variable suivante :

   ```
   byte yBroLED1 = 11 ;
   ```

4. Insère la même déclaration pour la seconde LED :

```
byte yBroLED2 = 10 ;
```

Les noms des variables ont été choisis ainsi :

y pour informer que le type est byte ;

Bro comme abréviation de broche ;

LEDx pour identifier la broche.

Tu découvriras dans le chapitre suivant pourquoi j'utilise les broches dans le sens décroissant.

Nous pouvons maintenant profiter de ces noms de variables dans les appels de fonctions.

5. Dans la première fonction, modifie les appels à la fonction `pinMode()` pour qu'ils s'écrivent ainsi :

```
void setup() {
pinMode(yBroLED1, OUTPUT) ;
pinMode(yBroLED2, OUTPUT) ;
}
```

Fais un copier-coller de la première ligne pour insérer la seconde. Il n'y a qu'à remplacer le 1 par un 2 dans le nom de la variable.

6. Dans la fonction de boucle `loop()`, effectue la même modification aux deux endroits où nous citons le numéro de la broche.

```
digitalWrite(yBroLED1, HIGH) ;
digitalWrite(yBroLED2, LOW) ;
```

7. Insère une copie des deux appels à `digitalWrite()` pour contrôler la seconde broche. Le résultat doit correspondre au Listing 6.2.

Listing 6.2 : Version 2 du projet Luciole

```
/* ARDUIPROG_KIDZ
 *  CH06A_Luciole2
 *  OEN1710
 */
byte yBroLED1 = 11;
byte yBroLED2 = 10;

void setup() {
  pinMode(yBroLED1, OUTPUT);
  pinMode(yBroLED2, OUTPUT);
}
void loop() {
  digitalWrite(yBroLED1, HIGH);
  digitalWrite(yBroLED2, LOW);
  delay(1000);
  digitalWrite(yBroLED1, LOW);
  digitalWrite(yBroLED2, HIGH);
  delay(1000);
}
```

8. Ajoute une seconde LED à côté de la première (laisse quelques rangées d'espace) sauf que tu la branche à la broche DIGITAL 10 au lieu de 11.

9. Tu peux maintenant tester ton programme. Tu dois voir les LED s'allumer tour à tour.

Intéressons-nous maintenant à une nouvelle fonction qui permet de faire varier un peu le comportement de ton Arduino : la fonction de génération de nombres pseudo-aléatoires.

Allez à Thouars !

Parmi les fonctions prédéfinies installées en même temps que l'atelier Arduino, il y en a une qui permet de générer un nombre entre deux valeurs de telle manière qu'il ne puisse pas aisément être prévisible. Ce n'est pas véritablement un nombre au hasard, et c'est pourquoi on parle de nombre pseudo-aléatoire. Cela suffira grandement à nos premiers essais.

La fonction correspondante porte le nom anglais `random()`, qui signifie « hasard » en anglais.

Cette fonction est prévue pour renvoyer une valeur numérique entière ; cette valeur peut être très grande. Pour la récupérer, il nous faut donc déclarer une variable d'un type approprié. Il s'agit ici d'un nouveau type qui s'écrit long. Le type long occupe 4 octets, soit deux fois plus que le type int. Le type long permet de stocker en mémoire un nombre vraiment grand :

entre environ –2 milliards et +2 milliards.

Je te propose d'utiliser ce tirage au sort pour faire varier le délai d'attente de nos allumages et extinctions de LED (Figure 6.13).

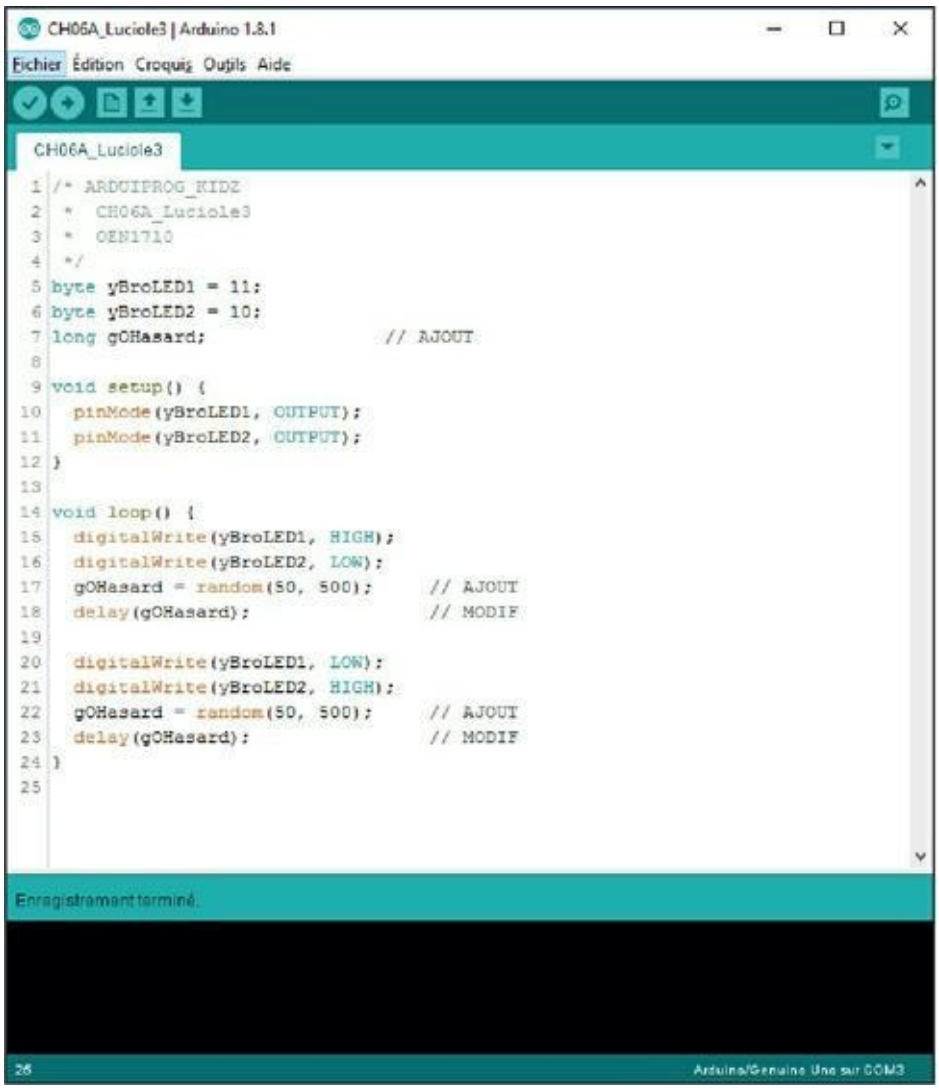


Figure 6.13 : Luciole3 dans l'éditeur.

1. En début de texte source, place-toi juste après la deuxième déclaration de variable et insère une ligne pour ajouter la déclaration suivante :

   ```
   long gOHasard ;
   ```

   Tu peux en profiter pour enregistrer cette troisième version du projet sous un nouveau nom.

2. Descends dans la fonction principale de boucle et place-toi avant le premier appel à la fonction `delay()`. Insère cette ligne :

   ```
   gOHasard = random(50, 500) ;
   ```

3. Dans le premier appel à la fonction de mise en pause, remplace la valeur numérique (c'est une valeur dite *littérale*) par le nom de la variable qui vient de recevoir la valeur tirée au sort.

4. Répète les étapes 2 et 3 pour la seconde LED.

   Le résultat doit correspondre au Listing 6.3.

Listing 6.3 : Troisième version du projet Luciole

```
/* ARDUIPROG_KIDZ
 *  CH06A_Luciole3
 *  OEN1710
 */
byte yBroLED1 = 11;
byte yBroLED2 = 10;
long gOHasard;

void setup() {
  pinMode(yBroLED1, OUTPUT);
  pinMode(yBroLED2, OUTPUT);
}

void loop() {
  digitalWrite(yBroLED1, HIGH);
  digitalWrite(yBroLED2, LOW);
```

```
  gOHasard = random(50, 500);
  delay(gOHasard);

  digitalWrite(yBroLED1, LOW);
  digitalWrite(yBroLED2, HIGH);
  gOHasard = random(50, 500);
  delay(gOHasard);
}
```

Lance un test du projet. Dorénavant, le clignotement devient imprévisible.

Tu as remarqué que les deux bornes de la plage de nombres parmi lesquels sera réalisé le tirage au sort, 50 à 500, sont telles que la valeur ne pourra jamais dépasser 499. Autrement dit, prévoir un type long est une gabegie. Mais la fonction est ainsi capable de produire une valeur bien plus grande.

D'ailleurs, voici une version bonus du projet dans lequel le tirage au sort est réalisé dans une sous-fonction qui renvoie une valeur après l'avoir rapetissée au format du type int, donc sur 2 octets.

Listing 6.4 : Version de test du projet Luciole

```
/* ARDUIPROG_KIDZ
 *  CH06A_Luciole4
 *  OEN1710
 */
byte yBroLED1 = 11;
byte yBroLED2 = 10;
long gOHasard;

void setup() {
```

```
  pinMode(yBroLED1, OUTPUT);
  pinMode(yBroLED2, OUTPUT);
  Serial.begin(9600);           // Pour
tester
  Serial.print("Largeur en octets du type
long: ");
  Serial.println( sizeof(gOHasard) );
}

void loop() {
  digitalWrite(yBroLED1, HIGH);
  digitalWrite(yBroLED2, LOW);
  delay( hasarder() );

  digitalWrite(yBroLED1, LOW);
  digitalWrite(yBroLED2, HIGH);
  delay( hasarder() );
}

int hasarder() {
  gOHasard = random(50, 500);
  Serial.println( gOHasard );

  return( int(gOHasard) );
}
```

Tout en bas, nous trouvons la fonction `hasarder()`. Observe bien la dernière instruction :

```
return( int(gOHasard) );
```

La mention `int(valeur)` force la valeur renvoyée à n'occuper que 2 octets. C'est d'ailleurs indispensable puisque nous avons déclaré dans la ligne de tête de la fonction qu'elle renverrait une valeur du type int.

Bien sûr, si la valeur dépasse les possibilités de ce type moins long, il y aura de la perte. Ceci dit, le compilateur Arduino n'est pas aussi sévère que d'autres compilateurs de langage C à ce niveau. Fais des essais en changeant le type pour le confirmer.

J'aurais pu écrire tout le contenu de la fonction `hasarder()` en une seule ligne :

```
return( int( random(50, 500) ) );
```

Il faut bien recompter le nombre de parenthèses ouvrantes et fermantes !

L'autre point remarquable est l'appel à `delay()` :

```
delay( hasarder() );
```

Comme durée de pause, nous faisons un appel à la fonction de tirage au sort. Et ça marche !

Récapitulons

Dans ce chapitre, nous avons vu :

- ce qu'est une diode LED ;
- ce qu'est une résistance et comment calculer et identifier sa valeur ;
- comment monter des LED sur des broches de sortie numériques ;

- la fonction de tirage au sort `random()` ;
- le transtypage avec `int()`.

Chapitre 7
Donner une impulsion

AU MENU DE CE CHAPITRE :

- **Sorties analogiques PWM**
- **Changement d'état logique**
- **Les valeurs binaires**
- **Tests imbriqués et branche else**
- **Le hasard fait bien les choses**
- **Un jeu de devinette numérique**

Dans le chapitre précédent, nous avons appris à utiliser les broches de sortie du mikon dans leur mode normal, c'est-à-dire en les faisant passer de 1 à 0 et de 0 à 1, sans autre possibilité.

Parfois, il faut fournir une valeur analogique en sortie, donc une tension qui progresse par petits pas entre 0 V et 5 V. Cela suppose l'existence d'un traducteur, un peu comme celui qui est utilisé par le mikon pour ses entrées analogiques. Pour une sortie, il s'agirait d'un convertisseur numérique vers analogique. Il fait le travail inverse du CAN (convertisseur analogique-numérique) vu dans le Chapitre 5.

Pour maintenir une très faible consommation, le mikon de la carte Arduino Uno (un AT328) n'est pas équipé d'un tel convertisseur numérique vers analogique (CNA ou DAC). Certains autres modèles de la gamme Arduino en possèdent, mais ils sont plus chers.

Tout n’est pas perdu ! Tu vas pouvoir profiter d’une technique qui permet de simuler une sortie analogique en faisant varier graduellement la proportion de temps à l’état 1 plutôt qu’à l’état 0 d’une broche de sortie. Cette technique est appelée la modulation de largeur d’impulsion PWM (Pulse Width Modulation) ou bien MLI (modulation de largeur d’impulsion).

La modulation de largeur d’impulsion

Le principe consiste à produire un signal carré, c’est-à-dire un signal qui passe par exemple 500 fois par seconde de 0 à 1 (500 Hz). Visuellement, cela ressemble aux créneaux en haut des murs des châteaux forts (les mâchicoulis). Le jeu consiste à déformer les carrés (Figure 7.1).

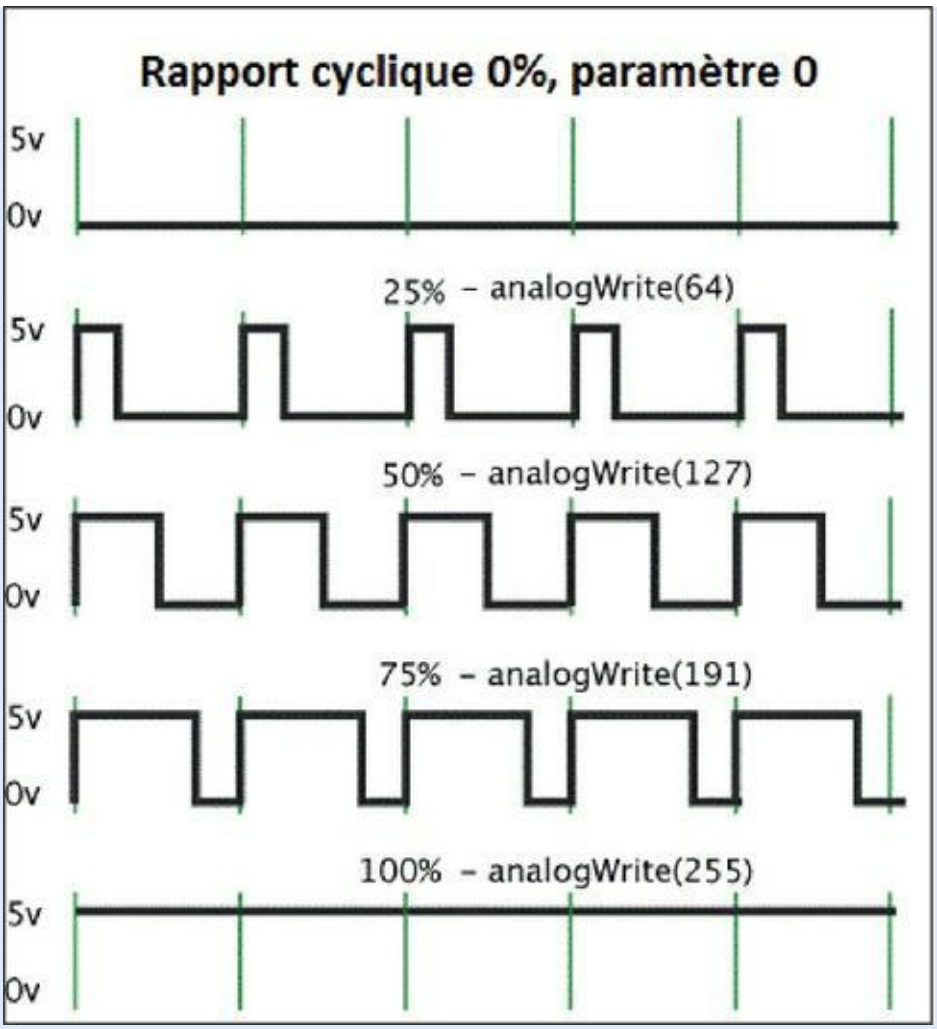


Figure 7.1 : Principe de la modulation de largeur d'impulsion PWM.

Pour envoyer sur la broche deux fois moins d'énergie que si elle restait à 1 tout le temps (comme en numérique), il suffit de rendre égales la durée à 0 et la durée à 1. On dit que le rapport cyclique est de 50 %. Pendant la moitié du temps, la sortie est éteinte (0 V).

Avec une fréquence de 500 Hz, la sortie est à 0 V pendant 1 milliseconde puis à 5 V pendant 1 milliseconde.

L'astuce consiste à faire varier la durée relative du 0 et du 1. Pour simuler une tension analogique de 1,25 V, je réduis la proportion

de temps à l'état 1 à 25 %. Si j'applique ce genre de signal à une diode LED, elle brillera plus ou moins fort selon le pourcentage de temps pendant lequel elle sera alimentée en énergie.

La fonction analogRead()

La fonction `analogRead()` s'utilise presque de la même manière que sa collègue `digitalRead()` vue dans le chapitre précédent. Elle attend deux paramètres. Voici sa syntaxe générale :

```
analogRead(numéro_broche,
valeur_entre_0_et_255);
```

La seule différence concerne le deuxième paramètre. Au lieu d'indiquer un des deux mots réservés HIGH ou LOW (que tu peux remplacer par 1 ou 0), tu indiques une valeur numérique entière entre 0 et 255.

Ce n'est pas la valeur numérique entre 0 et 255 qui est transmise sur la broche de sortie. Le mikon utilise cette valeur pour calculer un rapport cyclique entre 0 % et 100 %. Voici quelques étapes de l'équivalence :

Tableau 7.1 : analogRead() et rapport cyclique PWM

Valeur du paramètre	*Rapport cyclique*
0	0 %
64	25 %
128	50 %
192	75 %
255	100 %

Passons à la pratique. Nous allons réutiliser le montage de la fin du précédent chapitre, celui qui comportait deux diodes LED.

Au niveau du programme, nous allons également repartir d'une des versions du projet Luciole, la version `Luciole2`.

Le projet Binar

Nous allons réutiliser un de tes projets antérieurs. Si tu n'avais pas enregistré les différentes étapes, tu peux recharger le fichier d'exemple fourni avec le livre comme je l'explique dans les sections suivantes.

Montage du projet

Nous réutilisons sans aucune modification le circuit Luciole du chapitre précédent. Je t'invite à y revenir si tu as besoin de le reconstruire.

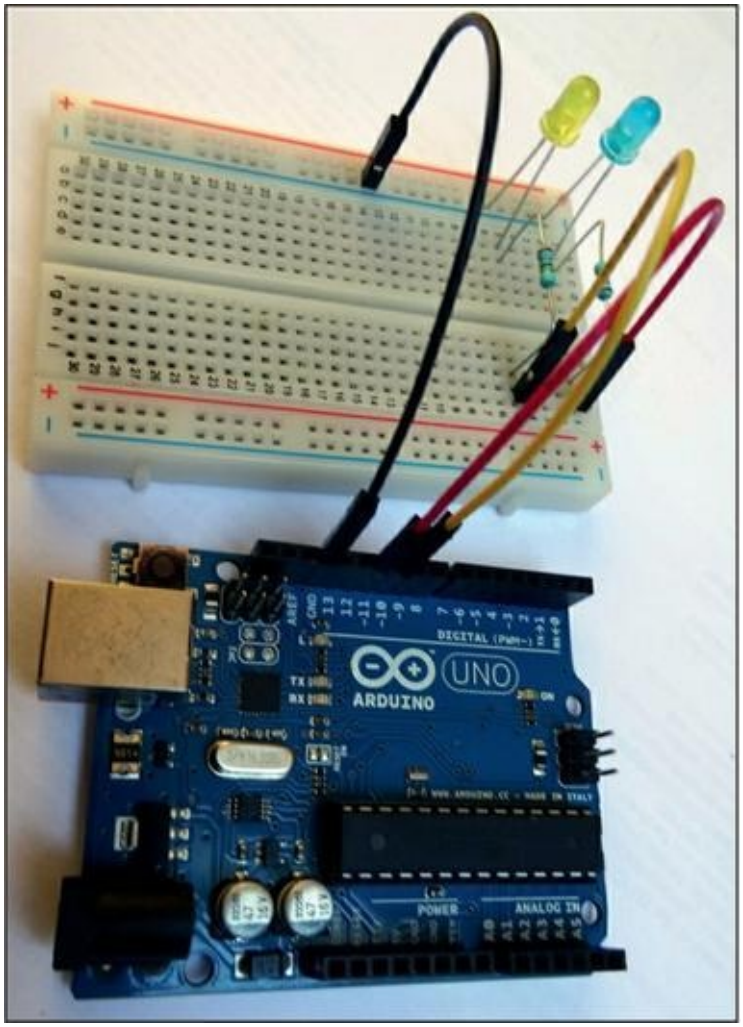


Figure 7.2 : Rappel du montage réutilisé du projet Luciole.

Recyclage de code

Voici comment procéder pour débuter le nouveau projet.

1. Démarre l'atelier Arduino si nécessaire, puis utilise la commande **Fichier/Ouvrir** pour charger le projet de la version 2 de Luciole. Dans mes exemples, il porte le nom suivant :

   `CH06A_Luciole2`

Si tu ne retrouves pas ton programme, ce n'est pas très grave : tu peux aussi ressaisir les quelques lignes du Listing 7.1 qui suit.

2. Procède immédiatement à l'enregistrement du projet sous un nouveau nom, pour ne pas modifier par mégarde le projet dont tu t'inspires. Voici le nom que j'ai choisi :

   `CH07A _Binar1`

3. Dans les commentaires de tête de fichier, modifie ce qui doit l'être, notamment le nom du projet.

4. Compare le Listing 7.1 à ce que tu vois à l'écran, ajoute toutes les nouvelles lignes, puis modifie les lignes selon les conseils que je donne un peu plus bas.

Listing 7.1 : Version 1 de Binar

```
/* ARDUIPROG_KIDZ
 *  CH07A_Binar1 (de CH06A_Luciole2)
 *  OEN1711
 */
byte yBroLED1 = 11;
byte yBroLED2 = 10;
byte yAnaOUI = 200;  // PWM
byte yAnaNON = 0;

void setup() {
  pinMode(yBroLED1, OUTPUT);
  pinMode(yBroLED2, OUTPUT);
```

```
}

void loop() {
  analogWrite(yBroLED1, yAnaOUI);
  analogWrite(yBroLED2, yAnaNON);
  delay(1000);
  analogWrite(yBroLED1, yAnaNON);
  analogWrite(yBroLED2, yAnaOUI);
  delay(1000);
}
```

La lecture de ce listing t'est certainement devenue beaucoup plus facile que celle de ton premier contact avec un programme !

Quelques explications s'imposent cependant.

Déclarations globales

Les deux variables globales pour déclarer les deux broches de sortie n'ont pas changé. Je les avais choisies en prévoyant de réutiliser le programme du chapitre précédent. En effet, toutes les broches numériques ne permettent pas de simuler une sortie analogique PWM. Sur la carte Arduino Uno, seules les broches suivantes sont utilisables :

```
3, 5, 6, 9, 10 et 11
```

Il n'y a donc que 6 broches sur 14 qui peuvent servir de sorties pseudo-analogiques.

Si tu utilises les broches 5 et 6 en tant que sorties analogiques PWM, tu remarqueras peut-être qu'il est impossible de régler ces sorties au minimum, c'est-à-dire à 0. C'est un petit défaut connu mais généralement sans inconvénient.

Nous déclarons ensuite deux nouvelles variables qui portent des noms qui peuvent t'étonner. Expliquons-nous.

La première lettre permet de symboliser le type byte. Comme la lettre b est déjà utilisée pour le type boolean, j'ai choisi la lettre y qui fait partie du nom byte. L'abréviation ana signifie évidemment « ANAlogique » . La fin de chacun des deux noms correspond à l'état haut (OUI) ou bas (NON).

Tu remarques que je donne comme valeur initiale à yAnaOUI la valeur 200. C'est un peu moins que la valeur maximale qui est égale à 255, mais cela suffit largement à faire briller les diodes LED.

Tu pourras tester le contrôle de l'intensité lumineuse des diodes LED en modifiant uniquement la valeur à cet endroit. En indiquant par exemple 10 au lieu de 200, les LED ne brilleront presque plus, voire pas du tout pour certaines.

Le fait que les LED s'allument encore, même avec une valeur de 10 ou de 20, prouve que ce n'est pas une tension analogique qui est fournie sur la sortie. Les diodes reçoivent toujours de temps à autre une petite impulsion à 5 V, mais si rarement qu'elles n'ont pas le temps de fournir beaucoup de lumière.

Préparation

Passons à la fonction de préparation setup(). Il n'y a aucune différence par rapport au projet Luciole, mais on peut faire une remarque : d'après les données du constructeur, il n'est pas obligatoire de préparer les broches par ces deux instructions si on veut les utiliser comme sorties analogiques. Mais puisqu'elle ne n'a pas d'effet négatif, je trouve qu'il vaut mieux conserver cette habitude.

Traitements

Passons à la fonction principale loop(). Les quatre appels à la fonction digitalRead() doivent être remplacés par la nouvelle fonction analogRead() et le second paramètre de chaque appel doit être remplacé par la valeur appropriée :

- HIGH est remplacé par yAnaOUI.
- LOW est remplacé par yAnaNON.

Figure 7.3 : La version 1 du projet prête à être téléversée.

Test de Binar1

Vérifie bien tes aménagements, puis lance un téléversement. Assure-toi qu'il n'y a pas de message orange dans le panneau inférieur.

Lorsque l'on fait des modifications progressives dans un texte source, on est tellement impatient que l'on se focalise sur leurs effets sur le montage au lieu de vérifier l'absence d'erreurs. Parfois, on est déçu parce que la dernière retouche ne semble avoir eu aucun effet. En fait (et cela m'est arrivé), une erreur dans le texte source empêche le nouveau téléversement ; la carte continue à exécuter l'ancienne version du projet. Te voilà averti.

Tu ne dois remarquer aucune différence avec le projet numérique Luciole. Si tu réduis la valeur de `yAnaOUI` à 5, les LED vont briller très faiblement ou plus du tout.

Les résistances utilisées peuvent être un peu trop faibles pour certains modèles de LED. Si tu constates qu'une LED reste brillante avec une valeur très faible, remplace sa résistance par une résistance de 470 ohms.

Jusqu'à présent, les choses étaient simples. Passons à la vitesse supérieure. Nous allons introduire des variables d'état et des tests conditionnels. Tu sauras tout, à condition de poursuivre.

Soyons logiques (Binar2)

Dans cette deuxième version, nous allons abandonner la technique élémentaire consistant à allumer une diode LED quand on en éteint une autre. Cette fois-ci, nous allons utiliser une variable d'état qui pourra être égale à 0 ou à 1. Nous pourrons ainsi décider d'allumer ou d'éteindre la seconde LED en fonction d'un test de cette variable d'état.

Nous allons donc déclarer une variable comme ceci :

```
boolean bEtatU = 0;
```

Je propose ce nom parce que c'est une variable du type booléen, que c'est une variable d'état et qu'elle concerne ce qui va être le chiffre de l'unité d'un projet de compteur binaire.

Dans la boucle principale, nous allons pouvoir tester cette variable et inverser son état à chaque tour de boucle. Mais voyons

directement cela dans le texte source.

Listing 7.2 : Version 2 de Binar

```
/* ARDUIPROG_KIDZ
 *  CH07A_Binar2
 *  OEN1711
 */
byte yBroLED1 = 11;
byte yBroLED2 = 10;
byte yAnaOUI = 100;
byte yAnaNON =   0;

boolean bEtatU = 0;                // AJOUT

void setup() {
  pinMode(yBroLED1, OUTPUT);
  pinMode(yBroLED2, OUTPUT);
  for (int i = 0; i < 100; i++) {
    analogWrite(yBroLED1, i); delay(20);
  }
  for (int i = 100; i > 0; i--) {
    analogWrite(yBroLED1, i); delay(20);
  }
}
void loop() {
  if (bEtatU == 0) {               //
Comparaison
    analogWrite(yBroLED1, yAnaOUI);
  }
  else {                           // SINON
    analogWrite(yBroLED1, yAnaNON);
```

```
  }

  bEtatU = !bEtatU;                    //
Inverseur

  if (bEtatU) {
    analogWrite(yBroLED2, yAnaNON);
  }
  else {
    analogWrite(yBroLED2, yAnaOUI);
  }
  delay(1000);
}
```

Saisis ce texte source en repartant de la première version que tu enregistres sous le nom `Binar2`.

Tiens bien compte de la différence entre majuscules et minuscules dans les noms des variables.

Dans la fonction `setup()`, j'ai ajouté deux boucles de répétition pour montrer le contrôle que nous avons dorénavant sur la luminosité de la première LED.

La fonction principale `loop()` a été entièrement remaniée. Voyons d'abord les six premières lignes de la boucle principale :

```
  if (bEtatU == 0) {                  //
Comparaison
    analogWrite(yBroLED1, yAnaOUI);
  }
  else {                               // SINON
    analogWrite(yBroLED1, yAnaNON);
```

```
  }
```

Voici le modèle de ce bloc conditionnel :

```
if (condition) {
  instruction1;
  instruction2;
}
else {
  instruction1;
  instruction2;
}
```

Juste après le mot if, nous indiquons entre parenthèses une condition sous la forme d'une expression. Il s'agit en fait d'une question à laquelle il sera répondu pendant l'exécution. Ici, nous cherchons à savoir si `bEtatU` est égal à 0, ou pas. Note bien le fait qu'il y a deux signes == qui se suivent. Ce n'est pas une erreur.

Un seul signe = est une opération de copie. Deux signes == forment un opérateur de comparaison.

Si tu n'avais indiqué qu'un seul signe = dans l'expression, le programme n'aurait rien trouvé à y redire, mais la première diode LED resterait allumée en permanence parce que l'expression est une instruction de copie de la valeur 0 dans la variable `bEtatU`. Et cette opération de copie est toujours vraie (heureusement !), donc égale à 1.

S'il est vrai que `bEtatU` vaut 0 à ce moment, le contenu des deux accolades du if est exécuté. Ici, nous allumons la première diode LED.

Si la variable d'état ne vaut pas 0, l'instruction du bloc if n'est pas exécutée. En revanche, puisque nous avons prévu une deuxième

branche qui commence par else (sinon), celle-ci sera exécutée, ce qui fera éteindre la première diode LED.

Une fois que la LED1 a été allumée ou éteinte, nous réglons le prochain état que devra prendre la LED au tour suivant. L'instruction est étonnante :

```
bEtatU =  !bEtatU;
```

Il n'y a qu'un signe =, donc c'est une copie de la valeur de droite dans la variable de gauche. Mais qu'est-ce que ce petit signe point d'exclamation ? Un détail ? Non, ce petit signe ! a un effet énorme : il inverse la valeur. Si la variable vaut 1, elle vaut ensuite 0 après cette autocopie inverseuse !

Nous arrivons enfin au deuxième bloc conditionnel, construit sur le même schéma que le premier. Cependant, l'expression est bizarre. Elle ne compare plus rien. C'est une technique autorisée : il suffit que la variable citée possède une valeur différente de 0 pour que l'expression soit considérée comme vraie, et que le teste réussisse. Ici, si la variable d'état vaut 1, nous éteignons la seconde diode LED. Sinon, nous l'allumons.

Tu remarques qu'il n'y a pas de signe point-virgule après la parenthèse droite de l'expression de test du if. De nombreux débutants font l'erreur d'ajouter un signe ; à cet endroit.

Des tests compacts sans accolades

Il faut un certain temps pour s'habituer à toutes ces nouvelles accolades, mais elles sont indispensables, sauf dans un cas particulier : lorsqu'il n'y a qu'une seule instruction dans une branche conditionnelle, les accolades sont superflues. Ceci dit, il est conseillé de les prévoir même dans ce cas : si tu ajoutes plus tard une seconde instruction sans délimiter un bloc, l'erreur sera

difficile à repérer. Seule la première sera exécutée de manière conditionnelle. La nouvelle le sera dans tous les cas.

Le Listing 7.3 montre exactement la fonction `loop()` du même programme, mais plus compact. J'ai ajouté des espaces dans les branches `else` pour rendre le résultat plus esthétique et plus lisible. Les espaces ne sont pas prises en compte à cet endroit.

Listing 7.3 : Fonction loop() compacte de Binar2

```
void loop() {
  if (!bEtatU) analogWrite(yBroLED1,
yAnaOUI);
          else analogWrite(yBroLED1,
yAnaNON);

  bEtatU =  !bEtatU;

  if (bEtatU) analogWrite(yBroLED2,
yAnaNON);
         else analogWrite(yBroLED2,
yAnaOUI);
  delay(1000);
}
```

Conclusion

Apparemment, nous en sommes toujours au même point : les deux diodes LED clignotent en alternance. Pourtant, la raison pour laquelle la seconde LED s'allume est devenue plus intelligente. Son état change en fonction d'une variable qui contrôle également l'état de l'autre diode LED. Nous allons pousser ce raisonnement un peu

plus loin dans la prochaine version du projet. Nous allons ainsi visualiser le comptage binaire.

Un plus un égale zéro ?

Tu as appris à compter selon le système décimal. Il est presque universel, mais ne l'a pas toujours été : les enfants mayas apprenaient à compter avec les dix doigts de leurs deux mains et les dix doigts de leurs deux pieds. Ils comptaient donc en base 20. Mais que signifie compter en base 10 comme nous le faisons sans réfléchir ?

En décimal, tu te sers des chiffres 0 à 9. Une fois arrivé à 9, tu retombes à 0, mais tu utilises un deuxième chiffre pour représenter la première dizaine. Une fois arrivé à 99, tu retombes à 0 dans les unités et dans les dizaines ; tu écris un premier chiffre dans la position des centaines.

Ces règles de numérotation sont exactement les mêmes pour le mikon et pour tous les ordinateurs actuels dans le monde. La seule différence est qu'il ne possède que deux chiffres pour compter : le 0 et le 1. Arrivé à 1, on retombe déjà à 0, avec une retenue qui devient une deuzaine.

Tu as peut-être déjà vu le compteur kilométrique d'une voiture un peu ancienne. Il est constitué de plusieurs petites roues portant les chiffres 0 à 9. À la hauteur du 9, un ergot dépasse afin d'entraîner d'un cran la roue située à sa gauche au passage du 9 vers le 0. On appelle ce compteur un odomètre.

Figure 7.4 : Odomètre d'un véhicule automobile.

En numérotation binaire, on ne parle pas de dizaines et de centaines, mais de deuzaine, de quatraine, de huitaine, de seizaine, etc. Et on en parle au singulier il ne peut y avoir qu'une seule deuzaine, une seule quatraine, etc.

À chaque chiffre binaire correspond un bit, mais tous les bits n'ont pas la même importance. En numérotation décimale, le chiffre des centaines vaut 100 fois plus que le chiffre des unités. En numérotation binaire, le chiffre de la quatraine vaut quatre fois plus que le chiffre de l'unité. Le Tableau 7.2 compare les deux bases :

Tableau 7.2 : Comparaison entre numérations décimale et binaire

Position du chiffre	4	3	2	1
En décimal	millier	centaine	dizaine	unité
Valeur décimale	1000	100	10	1

En binaire	huitaine	quatraine	deuzaine	unité
Valeur décimale	8	4	2	1

Voici comment compter en binaire de 0 à 8 (Tableau 7.3) :

Tableau 7.3 : Numération de 0 à 8 en binaire

Décimal	*Binaire*
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000

LSB et MSB

Le bit de l'unité, celui situé le plus à droite, est celui dont le changement a le moins d'effet sur la valeur globale. C'est donc le bit le moins significatif. Pour le désigner, on utilise en général le sigle anglais correspondant, *LSB*, qui signifie *Least Significant Bit*.

Inversement, le chiffre le plus à gauche est celui qui a le plus d'effet sur la valeur. C'est le bit le plus significatif, et on l'appelle *bit MSB* (*Most Significant Bit*).

Nous allons commencer par voir comment faire compter notre programme en binaire de 0 à 3. D'après le Tableau 7.3, on voit que deux bits suffiront. Cela tombe bien, puisque nous avons déjà deux diodes LED sur notre montage. Analysons les états successifs des LED (Tableau 7.4) :

Tableau 7.4 : Quelle LED allumer et éteindre ?

En décimal	*Diode MSB (la 2)*	*Diode LSB (la 1)*
Zéro	0	0
Un	0	1
Deux	1	0
Trois	1	1
Zéro	0	0
etc.		

Voici le raisonnement : la LED1 de l'unité s'allume et s'éteint alternativement. La LED des deuzaines doit changer d'état (s'allumer ou s'éteindre) lorsque la LED1 de l'unité retombe à l'état bas (à 0), c'est-à-dire lorsqu'elle s'éteint. Et seulement dans ce cas. C'est donc une fois sur deux.

L'astuce consiste à prévoir une variable dans laquelle ton programme se souviendra de l'état précédent de la diode LED1 de l'unité. Il ne faudra changer l'état de la diode LED2 de la deuzaine que lorsque les deux conditions suivantes seront réunies :

» la diode LED1 de l'unité est éteinte ET

» la diode LED1 de l'unité était allumée au tour précédent.

Bien sûr, la LED1 est tour à tour allumée et éteinte quand on compte en binaire. Mais en raisonnant comme je le propose ici, nous pourrons appliquer le même algorithme pour contrôler les chiffres binaires suivants : la quatraine avec l'état de la deuzaine, la huitaine avec celui de la quatraine, etc.

Un petit algorithme

Avant de passer à la rédaction du code, il est toujours conseillé d'écrire les grandes lignes du fonctionnement prévu.

1. Il nous faut ajouter une variable pour mémoriser l'état précédent celui en cours pour la diode LED1 de l'unité.
2. Il faut ajouter une instruction pour mémoriser l'état courant dans cette nouvelle variable pour l'état antérieur avant de modifier l'état courant.
3. Il faut enfin ajouter une instruction conditionnelle contenant deux expressions de test pour décider s'il faut inverser ou non l'état de la diode LED2 de la deuzaine.

On peut compter sur lui (Binar3)

Nous repartons de la dernière version de notre projet, `Binar2`.

1. Ouvre le fichier du projet `Binar2` et réalise immédiatement une copie sous un nouveau nom, par

exemple `Binar3`.

2. Dans la section initiale des déclarations, ajoute les deux déclarations de variables suivantes :

   ```
   boolean bPrecU = 0 ;

   boolean bEtatD = 0 ;
   ```

   La lettre U signifie Unité et la lettre D, Deuzaine.

3. Au niveau de la fonction de préparation `setup()`, tu peux supprimer les deux instructions qui faisaient une animation d'accueil sur la LED1.

4. Au début de la fonction principale `loop()`, ajoute une instruction pour mémoriser dans notre nouvelle variable `bPrecU` le contenu actuel de la variable d'état de l'unité. Cela deviendra l'état précédent :

   ```
   bPrecU = bEtatU ;
   ```

5. Place-toi juste avant le deuxième test conditionnel et insère le test suivant en faisant bien attention au nouveau symbole `&&` :

   ```
   if ( ( ! bEtatU) && (bPrecU) ) bEtatD = ! bEtatD ;
   ```

   La partie nouvelle est l'expression de test :

   ```
   ( ( ! bEtatU) && (bPrecU) )
   ```

   Il y a en fait deux expressions qui sont évaluées chacune à part, puis le résultat (vrai ou faux) de

chacune est combiné grâce au nouvel opérateur logique &&.

Il y a quatre possibilités. Une seule fait réussir le test et donc inverser la valeur de `bETatD` : les deux sous-expressions doivent être vraies (valeur différente de 0). Observe le Tableau 7.5 qui est une table de vérité :

Tableau 7.5 : Table de vérité du test

Valeur bEtatU	*Valeur bPrecU*	*Test réussi ?*
0	0	Non
0	1	OUI
1	0	Non
1	1	Non

On peut formuler cette double condition ainsi. Attention : il y a le signe inverseur ! au début du nom de la variable :

```
SI l'état actuel de l'unité n'est PAS vrai
(son inverse est vrai)

ET

SI l'état précédent de l'unité EST vrai

ALORS Inverser l'état de la variable d'état
de la deuzaine.
```

(sinon ne rien faire)

Ce qui revient à dire ceci :

```
SI la LED 1 est éteinte

ET

SI la LED 1 était allumée au tour précédent

ALORS nous exécutons l'instruction
d'inversion.
```

Autrement dit, pour que le test échoue, il suffit que l'état actuel de la diode LED de l'unité soit d'être allumée ou bien il suffit qu'elle ait déjà été éteinte dans le tour précédent.

Tu remarques les espaces que j'ai ajoutées autour des deux sous-expressions ; cela facilite la lecture.

L'écriture `! bEtatU` est déroutante au début. Cette expression est vraie lorsque l'inverse est faux. La variable doit donc contenir la valeur 0 pour que l'expression vaille 1 par inversion.

Il ne reste plus qu'à comparer ce que tu as saisi au Listing 7.4.

Listing 7.4 : Version 3 de Binar

```
/* ARDUIPROG_KIDZ
 *  CH07A_Binar3
 *  OEN1711
 */
byte yBroLED1 = 11;
byte yBroLED2 = 10;
byte yAnaOUI = 200;
byte yAnaNON = 0;
```

```
boolean bEtatU = 0;
boolean bPrecU = 0;      // AJOUT
boolean bEtatD = 0;      // AJOUT (D pour
deuzaine)
void setup() {
  pinMode(yBroLED1, OUTPUT);
  pinMode(yBroLED2, OUTPUT);
  // Plus d'animation
}

void loop() {
  bPrecU = bEtatU;
// AJOUT

  if ( !bEtatU ) analogWrite(yBroLED1,
yAnaOUI);  // MODIF
            else analogWrite(yBroLED1,
yAnaNON);

  bEtatU =  !bEtatU;

  if ( (!bEtatU) && (bPrecU) ) bEtatD =
!bEtatD;   // AJOUT

  if ( bEtatD ) analogWrite(yBroLED2,
yAnaOUI);   // MODIF
           else analogWrite(yBroLED2,
yAnaNON);
  delay(1000);
}
```

Test de Binar3

Comme d'habitude, branche si nécessaire ta carte Arduino, téléverse le programme, corrige-le s'il le faut, puis admire le résultat. Les lettres clignotent en comptant de 0 à 3 en numérotation binaire.

Si tu enlèves la pause `delay()`, toutes les LED vont sembler allumées en permanence, tellement le programme va vite.

Montons en puissance en faisant compter de 0 à 15. Cela suppose d'installer deux autres diodes LED.

Un quartet en binaire

Sais-tu comment on appelle la moitié d'un octet ? Tout simplement un quartet. C'est ce que nous allons pouvoir contrôler dans cette dernière étape du projet.

Montage de Binar4

Pour le montage, il suffit de dupliquer les deux circuits de LED existants pour en avoir quatre au total :

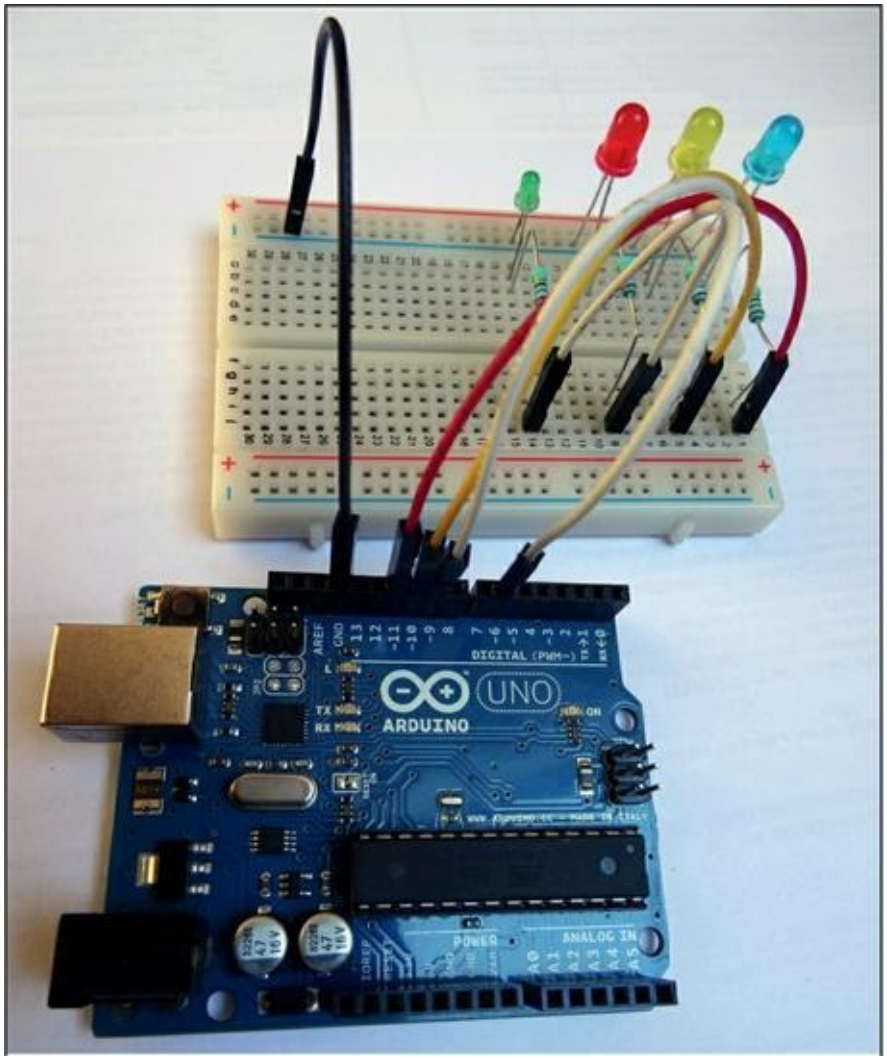


Figure 7.5 : Montage avec quatre diodes LED.

1. Branche sur la plaque d'essai une troisième et une quatrième diode LED.

   Si tu branches une LED à l'envers, elle ne s'allumera pas, mais cela ne fera pas de dégâts. Distingue bien anode et cathode.

2. Ajoute une résistance à chacune.

3. Branche un strap entre la résistance et deux autres broches de sortie numériques :

   pour la troisième LED de quatraine, branche-toi sur la broche 9 ;

   pour la dernière LED de huitaine, branche-toi sur la broche 6.

   Je rappelle que toutes les broches numériques ne permettent pas de travailler en pseudo-analogique PWM.

4. Vérifie ton montage.

C'est tout ce qu'il y a à faire. Passons au codage.

Texte source de Binar4

Voici maintenant le texte source de la quatrième et dernière version du projet Binar. Lis-le d'abord. Mes explications sont à la suite.

Listing 7.5 : Version 4 de Binar

```
/* ARDUIPROG_KIDZ
 *  CH07A_Binar4
 *  OEN1711
 */
byte yBroLED1 = 11;   // U = Unite
byte yBroLED2 = 10;   // D = Deuzaine
byte yBroLED3 = 9;    // Q = Quatraine
byte yBroLED4 = 6;    // H = Huitaine

byte yAnaOUI = 100;
```

```
byte yAnaNON = 0;

boolean bEtatU, bEtatD, bEtatQ, bEtatH = 0;
boolean bPrecU, bPrecD, bPrecQ, bPrecH = 0;
// AJOUT

void setup() {
  pinMode(yBroLED1, OUTPUT);
  pinMode(yBroLED2, OUTPUT);
  pinMode(yBroLED3, OUTPUT);
  pinMode(yBroLED4, OUTPUT);
}

void loop() {
//LED UNITE
  bPrecU = bEtatU;
  if ( !bEtatU ) { bEtatU = 1; }
            else { bEtatU = 0; }
  agir(yBroLED1, bEtatU);
  delay(50);  // DELAI PEDAGOGIQUE

//LED DEUZAINE
  bPrecD = bEtatD;
  if ( (!bEtatU)  && (bPrecU) ) {
    bEtatD =  !bEtatD;
    agir(yBroLED2, bEtatD);
  }
  delay(50);

//LED QUATRAINE
  bPrecQ = bEtatQ;
  if ( (!bEtatD)  && (bPrecD) ) {
```

```
    bEtatQ =  !bEtatQ;
    agir(yBroLED3, bEtatQ);
  }
  delay(50);
//LED HUITAINE
  bPrecH = bEtatH;                // Utile si
extension 8 bits
  if ( (!bEtatQ)  && (bPrecQ) ) {
    bEtatH =  !bEtatH;
    agir(yBroLED4, bEtatH);
  }

  delay(500);
}

void agir(byte broche, boolean valeur) {
  if (valeur) analogWrite(broche, yAnaOUI);
  else        analogWrite(broche, yAnaNON);
}
```

Mon exemple occupe 60 lignes. Cela commence à être remarquable. On est loin de Blink. Quelques commentaires ne seront donc pas inutiles.

Les déclarations globales

Dans la tête du programme, nous trouvons deux déclarations pour les deux nouvelles diodes LED. Tu remarques qu'elles sont branchées sur les broches 9 et 6, qui sont capables de fournir un signal PWM.

Au sujet des variables d'état, nous ne faisons plus dans le détail : nous déclarons quatre booléens pour l'état actuel et quatre autres

pour l'état précédent.

On pourrait « pinailler » en faisant remarquer que nous déclarons `bPrecH`, mais qu'elle ne sert à rien. En effet, elle serait utile pour une cinquième LED. En guise d'exercice, tu peux créer une version à huit LED pour compter de 0 à 255.

La configuration initiale

Nous déclarons en mode sortie les deux nouvelles broches. Il n'y a rien d'autre à ajouter dans la préparation.

La boucle principale

Elle évolue beaucoup, en apparence. Nous trouvons quatre blocs, dont trois très similaires. Pour la LED de l'unité, nous ne cherchons pas l'état antérieur : nous nous contentons d'inverser l'état, ce que je fais de façon très explicite. Tu sais toi aussi écrire la même action de façon beaucoup plus compacte :

```
bEtatU =  !bEtatU;
```

La grande nouveauté est l'introduction d'un appel à une nouvelle fonction que j'ai décidé de nommer `agir()`. Elle va nous épargner de répéter les appels à `analogWrite()` dans chaque bloc d'instructions des diodes LED. Si tu descends en bas du texte source, tu trouves le corps de cette nouvelle fonction. Tu constates qu'elle est polyvalente : elle est appelée quatre fois, avec des valeurs de paramètres différentes, évidemment.

J'ajoute enfin une petite pause que tu peux enlever si tu le désires. Elle permet de voir la transition entre une LED et la LED suivante (qui vaut le double).

Dans les trois sous-blocs des LED de deuzaine, quatraine et huitaine, tu constates que le test réunit deux expressions, comme déjà vu dans la version précédente du projet. Note au passage que

nous n'appelons la fonction `agir()` que lorsqu'il faut changer d'état. Autrement dit, si la LED était allumée, elle le reste lorsque le test ne réussit pas.

Nous en avons maintenant terminé avec le projet Binar. Appliquons toutes ces connaissances pour créer un jeu de devinette binaire, `Devinum`.

Voici Maître Devinum

Ce projet est un jeu de société. Tu vas pouvoir réunir tes amis pour essayer de décoder le plus vite possible des valeurs binaires qui vont être présentées successivement sur quatre diodes LED, donc une valeur décimale entre 1 et 15 (le 0 n'est pas intéressant, puisque tout est éteint). Chacun note ses réponses sur un bout de papier.

Une fois les dix valeurs affichées, le programme va montrer les bonnes réponses dans la fenêtre du Moniteur série. Vous comptez le nombre de bonnes réponses et tenez vos scores.

Les dix valeurs vont être tirées au sort. Nous exploitons à cet effet la fonction `random()`.

Pour nous épargner de devoir déclarer dix variables et de les manipuler l'une après l'autre, nous allons profiter d'une technique extrêmement importante en programmation : le tableau de valeurs.

Le tableau : variables en familles

Un tableau est une variable collective. Il porte un nom et possède un type, comme les variables simples. La différence est qu'il permet de regrouper plusieurs variables du même type. Pour accéder à chacune d'elles, il faut utiliser une notation spéciale avec la position de la variable par rapport au début du tableau (un indice). Voici l'écriture générique de la déclaration d'un tableau :

```
type  nomTableau[nombre_d'éléments];
```

Voici comment déclarer un tableau de dix entiers :

```
int nomTableau[10];
```

Et voici comment stocker une valeur dans la troisième case de ce tableau :

```
nomTableau[2] = 255;
```

J'ai bien écrit « dans la troisième case » , pas « dans la deuxième » . L'indice entre crochets désigne la case dans laquelle on veut écrire ou depuis laquelle on veut lire, mais en commençant la numérotation à 0. La dixième case correspond donc à l'indice 9 et la première case, à l'indice 0. C'est une des erreurs les plus fréquentes des débutants que de se tromper d'une case.

Les tableaux constituent un outil très puissant et on peut aisément s'y perdre. Dans mon autre livre, Programmer pour les Nuls, je montre un tableau à trois dimensions : un tableau qui contient un tableau dans chacune de ces cases, et ce tableau imbriqué contient lui-même un troisième tableau dans chaque case !

Bien sûr, avec la mémoire très limitée du mikon, déclarer un tableau à plusieurs dimensions va vite épuiser l'espace mémoire disponible.

Lecture directe d'un bit : bitRead()

L'autre nouveauté de notre projet est l'utilisation d'une fonction prédéfinie qui permet de tester directement la valeur d'un bit dans une valeur numérique. Il s'agit de la fonction `bitRead()`. Elle s'utilise de la manière suivante :

```
bitRead(valeur_à_scruter, rang_du_bit);
```

Le premier paramètre contient la valeur qui peut être sur 8, 16 ou 32 bits.

Le second paramètre désigne le numéro du bit, un peu comme un indice. D'ailleurs, la numérotation commence ici aussi à 0.

Par exemple, la valeur décimale 254 s'écrit ainsi en binaire :

```
1111 1110
```

Seul le bit de rang 0 (celui de droite) est à 0. Pour obtenir un test fondé sur sa valeur, on peut écrire ceci :

```
if ( bitRead(valeur, 0) )  { instructions si
le bit 0 est à 1; .... }
```

Par exemple, pour savoir si un nombre est pair ou impair, il suffit de tester son bit de rang 0 (le bit de poids faible, LSB, vu dans le projet précédent). Toutes les valeurs numériques binaires dont le bit de poids faible est à 1 sont nécessairement impaires. Tu l'as sans doute remarqué dans les allumages de LED du projet précédent.

Voyons maintenant le code source de `Devinum`. Les explications viennent à la suite.

Listing 7.6 : Texte source de Devinum

```
/* ARDUIPROG_KIDZ
 *  CH07B_Devinum
 *  OEN1711
*/

#define PAUSE 400
#define LUMIN 100

const byte yLed1 = 11;
```

```
const byte yLed2 = 10;
const byte yLed4 = 9;
const byte yLed8 = 6;

byte ayTablo[10];    // Le fameux tableau!

void setup() {
  pinMode(yLed1, OUTPUT);
  pinMode(yLed2, OUTPUT);
  pinMode(yLed4, OUTPUT);
  pinMode(yLed8, OUTPUT);
  Serial.begin(9600);
  Serial.println("*** Bienvenue chez DEVINUM
***");
}

void loop() {
  for (byte i = 0; i < 10; i++) {
    ayTablo[i] = random(1, 15);
    Serial.println(" ");
    Serial.print(ayTablo[i]);
    afficherBin(ayTablo[i]);
    toutEteindre();
  }
  afficherReponses();
  attendreSaisie();
}
void afficherBin(byte bNum) {
  if (bNum == 0) return;
  Serial.print("\t Binaire ");
  if (bitRead(bNum, 0)) agir(yLed1, LUMIN);
  else agir(yLed1, 0);
```

```
  if (bitRead(bNum, 1)) agir(yLed2, LUMIN);
  else agir(yLed2, 0);
  if (bitRead(bNum, 2)) agir(yLed4, LUMIN);
  else agir(yLed4, 0);
  if (bitRead(bNum, 3)) agir(yLed8, LUMIN);
  else agir(yLed8, 0);
  delay(PAUSE*3);
}

void agir(byte bBroche, byte bValeur) {
  analogWrite(bBroche, bValeur);
  Serial.print(boolean(bValeur));
}

void toutEteindre() {
  analogWrite(yLed8, 0);
  analogWrite(yLed4, 0);
  analogWrite(yLed2, 0);
  analogWrite(yLed1, 0);
  delay(PAUSE*3);
}

void afficherReponses() {
  delay(PAUSE*10);
  Serial.println();
  Serial.println("Maitre Devinum te donne
les solutions : ");

  for (byte i = 0; i < 10; i++) {
    Serial.print(ayTablo[i]);
    Serial.print(" ¦ ");
  }
```

```
  Serial.println();
  Serial.println("****************** ");
  Serial.println("Clic dans zone de saisie
en haut du Moniteur...");
  Serial.println("... puis Espace et Entree
pour autre partie !");

}
void attendreSaisie(){
  while ( Serial.available() == 0 ) ;
// Vigilance ici
  while ( Serial.available() > 0 )
Serial.read();
  Serial.flush();
}
```

Analyse du texte source

Quatre-vingts lignes. De plus en plus fort ! Tu remarques que le programme ressemble dans certaines parties à la dernière version de Binar. Pourtant, j'ai pris un malin plaisir à ne pas réutiliser les noms des variables. C'est volontaire. Il faut entraîner tes capacités d'adaptation. La logique est quasiment la même, mais les noms sont différents.

Pour créer ce texte source, tu peux néanmoins partir de `Binar4`, en enregistrant le projet sous le nom `Devinum`.

1. Démarre un nouveau projet (**Fichier/Nouveau**) ou réutilise `Binar4`.
2. Dans la section des déclarations globales, tu remarques deux nouvelles instructions qui

commencent par un signe # :

```
#define PAUSE 400

#define LUMIN 100
```

Il ne s'agit pas de variables mais simplement de symboles qui seront remplacés par la valeur numérique lorsque l'atelier va compiler le programme. Autrement dit, partout où tu vas citer le mot PAUSE (en lettres capitales, c'est une convention), ce mot sera remplacé par la valeur 400. L'énorme avantage est que cela permet de centraliser la valeur. Pour changer la vitesse d'exécution de toutes les phases du jeu en une seule fois, il te suffira de modifier la valeur numérique associée à PAUSE. Il en va de même pour la luminosité des diodes LED avec le mot LUMIN. Ce sont des sortes de synonymes.

3. La dernière déclaration globale correspond à notre fameux tableau. Nous l'avons déjà présenté plus haut.

   La fonction de préparation ne contient rien de particulier, mais tu notes un message de bienvenue qui s'affichera dans le Moniteur série.

   La boucle principale est étonnamment brève. Elle se résume quasiment à une grande boucle de répétition for. Toute la partie de questionnement du jeu est réunie à cet endroit.

4. Nous commençons par stocker dans chaque case du tableau la valeur renvoyée par la fonction qui réalise le

tirage au sort. Tu remarques que nous appelons la fonction `random()` avec une valeur entre 1 et 15. En effet, si nous avions choisi les limites 0 à 15, la fonction aurait parfois renvoyé la valeur 0, qui n'est pas très facile à deviner puisque rien n'est allumé.

Dans chaque tour de boucle, nous appelons d'abord une fonction pour afficher la valeur sur les quatre LED, puis une autre pour faire une transition en éteignant toutes les LED.

Une fois sortis de dix tours de boucle, nous affichons les réponses et attendons la saisie d'un caractère dans le Moniteur série pour lancer la partie suivante.

5. La fonction `afficherBin()` n'a besoin de recevoir en entrée que la valeur qu'elle doit afficher. C'est ici que nous utilisons la nouvelle fonction `bitRead()`. Tu remarques que nous appelons une autre fonction depuis cette fonction, la fonction `agir()`.

6. J'ai placé la définition de la fonction `agir()` juste après la précédente pour plus de confort. Que la valeur soit nulle ou non, nous réalisons deux actions : l'allumage ou l'extinction d'une diode LED et l'affichage d'un 1 ou d'un 0.

   La fonction `toutEteindre()` ne réclame aucun commentaire.

7. Dans la fonction `afficherReponse()`, nous présentons les dix valeurs qui sont toujours présentes

dans le tableau. Note au passage que le préfixe du nom du tableau est `ay` : `a` est la première lettre du mot anglais pour tableau, *array*. La lettre `y` rappelle qu'il s'agit d'un tableau de valeurs du type byte.

8. Nous arrivons à la dernière fonction, `attendreSaisie()`. Elle contient deux lignes étonnantes :

```
while ( Serial.available() == 0 ) ;
while ( Serial.available() > 0 )
Serial.read() ;
```

Ce genre d'écriture peut facilement devenir un trou noir, une boucle infinie. En effet, derrière la condition du mot while, donc après la dernière parenthèse fermante, il doit y avoir une instruction, ou même plusieurs si tu ajoutes une paire d'accolades pour faire un bloc. S'il n'y a que le signe point-virgule sans instruction, la boucle va se répéter tant que la condition entre les parenthèses reste vraie.

La fonction `Serial.available()` renvoie une valeur différente de 0 si quelque chose a été saisi dans le Moniteur série. Autrement dit, dans notre exemple, tant que la fonction `Serial.available()` renvoie la valeur 0, l'expression est vraie, et nous continuons à appeler sans cesse la même fonction en boucle. Cela permet d'attendre la saisie d'un caractère pour démarrer une autre partie.

Si aucun caractère n'est jamais saisi dans le Moniteur série, nous resterons dans cette boucle jusqu'à la fin du monde. Il faut donc être très attentif avec le signe point-virgule isolé dans les boucles de répétition.

Dans la seconde ligne, nous utilisons la même fonction, mais en sens inverse. Si nous avons atteint cette deuxième instruction, c'est que quelque chose a été saisi. Mais il faut continuer à lire les caractères un par un tant qu'il y en reste dans la mémoire du clavier.

Cette précaution permet d'éviter d'enchaîner deux parties successives lorsque l'on frappe par mégarde plusieurs caractères avant de valider par la touche Entrée.

Pour plus de précautions, nous ajoutons un appel à `Serial.flush()` pour vider la zone mémoire des caractères saisis. En anglais, flush signifie « purger » .

Et maintenant, on s'amuse !

N'hésite pas à tester le programme avec des amis. Parmi les améliorations que je te propose d'envisager, voici les plus évidentes :

- Pour rendre le jeu plus difficile, il suffit de réduire la durée de pause. Comme j'utilise des multiples de cette valeur initiale, le changement sera répercuté partout.
- Tu peux augmenter le nombre de chiffres à deviner, ce qui suppose d'effectuer plusieurs retouches. Il faut changer les dimensions du tableau dans les déclarations, et modifier l'indice dans la boucle for qui utilise ce tableau. Je rappelle au passage que l'indice de répétition commence à 0 pour qu'il puisse servir simultanément d'indice dans le tableau. La valeur 0 sert à lire la première case du tableau.
- Une amélioration bienvenue consiste à relancer le tirage au sort si la valeur renvoyée est la même que la précédente. Il est trop facile de faire deviner deux fois de suite le même nombre. Tu peux chercher la solution tout seul. Quelques indices : il faut une variable pour la valeur précédente et une nouvelle

> fonction qui lance un nouveau tirage tant qu'il y a doublon. La solution se trouve dans
>
> l'exemple `CH07B_Devinum2`.

Pour bien jouer à Devinum, placez-vous devant les LED de sorte que celle de l'unité soit à droite devant toi. C'est le LSB !

Récapitulons

Dans ce chapitre, nous avons vu :

- l'utilisation des broches numériques comme sorties analogiques grâce à la technique PWM ;
- les tests conditionnels et les boucles de répétition ;
- les tableaux de données ;
- l'utilisation du Moniteur série en affichage et en saisie.

Dans les prochains chapitres, nous allons mettre en pratique les connaissances décrites dans cette Semaine 2.

Semaine 3 : Son et lumière

Au menu de cette semaine :

Chapitre 8 : Ta baguette magique

Chapitre 9 : Que personne ne bouge !

Chapitre 10 : Le voleur de couleurs

Chapitre 8
Ta baguette magique

AU MENU DE CE CHAPITRE :

- **Un capteur de lumière invisible**
- **Une librairie avec des objets qui ont de la classe**
- **La gare de triage** switch case
- **Un tableau de tableaux**

Aimerais-tu avoir une baguette magique ? Tu pourrais ainsi commander toutes sortes d'équipements en envoyant des ordres de façon invisible. Je suis certain que tu penses déjà à la télécommande du téléviseur ou de la chaîne stéréo. Il y a tellement de boutons sur ce genre d'objet ! Il y a sans doute un code par touche, et un code numérique. Mais comment faire pour capturer ces codes afin d'influer sur un programme Arduino ?

Le principe est proche de celui qu'utilisent les marins pour communiquer entre deux bateaux : avec un phare dont on ouvre et ferme le couvercle pour l'allumer et l'éteindre. C'est donc un code binaire, constitué de 0 et de 1. Mais pour que la baguette soit vraiment magique, il faudrait que le rayon soit invisible.

Comme toutes les ondes, les ondes lumineuses vibrent à une certaine fréquence. L'œil humain détecte une gamme assez limitée de fréquences : ce sont les couleurs de la lumière. En dehors de cette plage, nous ne voyons pas les ondes, et pourtant elles existent.

Heureusement que nous ne voyons pas toutes les ondes électromagnétiques : stations de radio, radios des pompiers et des policiers, réseaux Wi-Fi, téléphones GSM, radars des avions et des

aéroports, etc. Nous serions aveuglés !

Hormis la plage visible, il y a deux possibilités : une fréquence plus grande ou plus petite. Les fréquences inférieures à celle de la plus basse fréquence détectée par un œil correspondent à moins rouge que le rouge foncé. Cela s'appelle l'infrarouge. Tout le monde possède au moins un exemplaire de l'émetteur d'ondes infrarouges qu'il nous faut : une télécommande (Figure 8.1).

Figure 8.1 : Une télécommande comme il en existe des millions.

Quand on appuie sur une touche, la télécommande émet environ 40 000 éclairs de lumière infrarouge par seconde. Ces éclairs représentent des suites de 0 et de 1 binaires. C'est ce qui permet de transmettre les codes numériques des touches de la télécommande.

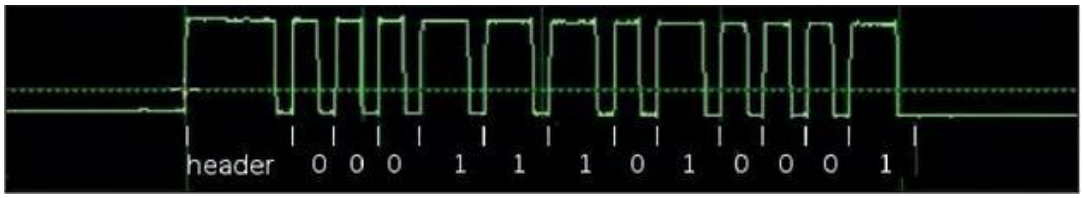


Figure 8.2 : Train d'ondes infrarouge d'une télécommande.

Par exemple, supposons que la touche de la première chaîne corresponde au code numérique 2500 et celui de la deuxième chaîne, au code 2600. Comment le récepteur du signal va-t-il

distinguer l'envoi de 2500 puis de 2600 de l'envoi en une fois de 25002600 ?

Le bon protocole

Imagine que tu rentres dans une boulangerie et que tu dises d'une traite ceci : « Bonjour, je voudrais une baguette, merci, au revoir. » Si la boulangère a suffisamment d'humour, elle répondra, pendant que tu refermes la porte : « Bonjour, oui bien sûr, 1,10 euro s'il vous plaît, bonne journée. » Ce n'est pas le bon protocole : il faut d'abord dire ta première phase, attendre la réponse de la boulangère, dire la seconde, etc.

Pour les codes d'une télécommande, un protocole détermine également la façon dont elle doit transmettre les valeurs. Les protocoles des télécommandes sont presque normalisés, mais pas totalement. Il y a plusieurs normes car plusieurs fabricants. C'est pour cette raison que l'on trouve dans le commerce des télécommandes dites universelles, c'est-à-dire qui peuvent s'adapter à une marque ou une autre.

Dans ses grandes lignes, le protocole définit d'abord des codes spéciaux pour dire qu'un code de touche va être envoyé, puis que l'envoi est terminé. Mais toute cette analyse va te demander un énorme travail de codage, non ?

Bonne nouvelle : d'autres passionnés d'Arduino ont déjà réfléchi à ce problème et ont écrit les fonctions permettant de bien recevoir les codes d'une télécommande infrarouge. Toutes ces fonctions ont été réunies dans un conteneur appelé librairie et que tu peux utiliser gratuitement grâce à l'esprit collaboratif Arduino.

Grâce à ces fonctions, tu peux directement récupérer les valeurs numériques des codes des touches qui ont été reçues. Mais au fait, reçues comment ?

Un capteur d'ondes IR

Bonne nouvelle : pour posséder ta baguette magique, il suffit de brancher sur la carte Arduino un capteur de lumière infrarouge qui ne vaut que 1 ou 2 euros et de récupérer une télécommande dont l'appareil commandé a rendu l'âme. Tu peux aussi emprunter une télécommande de la maison, car cela ne va pas l'endommager.

Tu en sais assez pour te lancer dans la création du projet. En pratique, chaque code reçu pourrait contrôler une broche de sortie numérique pour, par exemple, activer un relais, démarrer un moteur électrique, faire émettre un son sur une carte son, etc.

Mais ce livre a pour but premier de faire découvrir des bases solides en programmation. Je ne vais donc pas te demander de sortir le fer à souder. Nous allons nous servir des codes détectés pour sélectionner des membres de phrase, ce qui donnera une sorte de jeu de création littéraire.

Expression des besoins

Nous allons puiser dans un stock de mots, donc dans des variables. Pour que le résultat soit intéressant, il nous faut pouvoir choisir parmi trois possibilités pour chaque morceau de phrase (sujet, verbe, adverbe, complément), ce qui revient à déclarer douze variables.

Je te propose de faire quatre tours de sélection.

À chaque tour, tu tapes sur une des trois touches de télécommande que nous prévoyons de détecter (en ignorant toutes les autres).

- Au premier tour, tu sélectionnes un sujet parmi trois.
- Au deuxième tour, tu sélectionnes un verbe parmi trois.
- Au troisième tour, tu sélectionnes un adverbe parmi trois.

- Au dernier tour, tu sélectionnes un complément parmi trois.

Comme nous n'utilisons pas d'écran LCD pour l'affichage, le résultat sera montré dans la fenêtre du Moniteur série. Bien sûr, les choix peuvent aussi changer l'état de plusieurs broches de sortie numériques ou analogiques.

Le projet va être réalisé en trois étapes :

1. Création de la partie décodage des touches.
2. Ajout de la logique de sélection des actions en fonction de chaque touche.
3. Création d'un conteneur à données pour construire les phrases.

Commençons par la partie proche du matériel : la détection des codes de touches.

Montage du projet BagMagik

Le montage va être simplissime, puisque nous n'avons besoin d'installer que le capteur infrarouge puis de brancher ses trois fils. Il n'y a même pas de résistance à ajouter.

Voici tout de même la petite liste des composants à réunir :

- le capteur infrarouge de type TSOP38238 ;
- une plaque d'essai, même une toute petite ;
- ta carte Arduino avec son câble ;
- trois straps mâle/mâle, si possible un rouge, un noir et un jaune ou vert.

Identifions le composant capteur

En observant le capteur TSOP38238 de près, tu constates qu'il possède une face bombée, avec une vitre sombre, destinée à laisser entrer la lumière infrarouge. Voici l'aspect général de ce capteur (Figure 8.3).

Figure 8.3 : Le capteur infrarouge.

Pour le repérage des trois pattes du capteur, c'est très simple. Tiens le capteur entre deux doigts devant toi, en sorte de voir la face bombée :

1. La patte de gauche (N° 1) est la patte de sortie du signal.
2. La patte du milieu (N° 2) va vers la masse ou GND.
3. La patte de droite (N° 3) doit être branchée au pôle positif, le 5 V.

Nous pouvons passer au montage.

1. Insère le capteur dans trois rangées de la plaque d'essai, de manière à pouvoir ajouter un bout de strap

dans chacune des rangées (Figure 8.4).

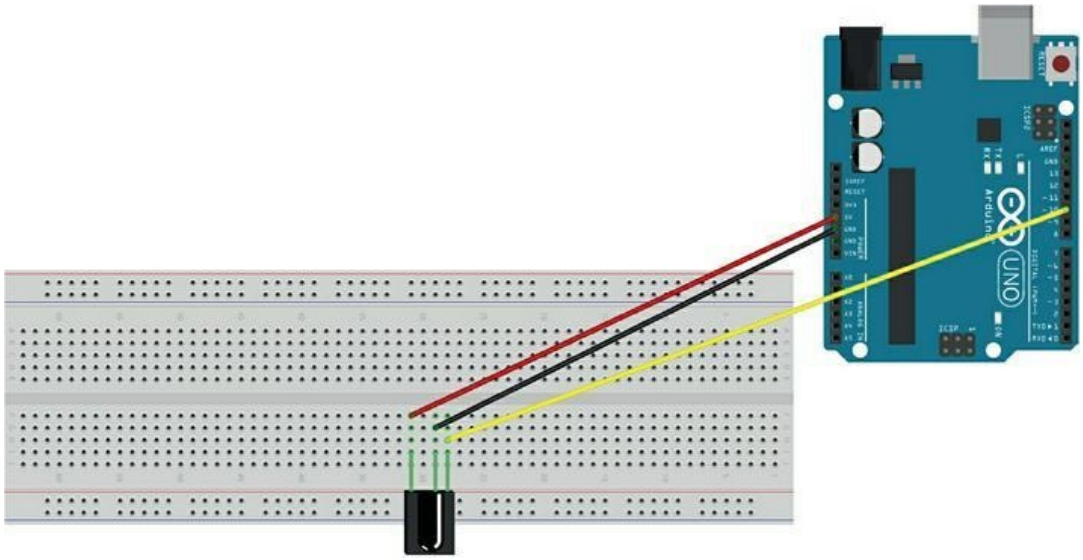


Figure 8.4 : Montage et branchement du capteur infrarouge.

2. Insère un strap noir dans la rangée de la patte du milieu et l'autre bout dans une des deux broches GND du connecteur en bas à gauche de la carte Arduino.
3. Branche le strap (jaune si possible) du signal entre la rangée de la première patte du capteur et la broche DIGITAL 10 dans le connecteur du haut de la carte.
4. Branche enfin le fil rouge entre la rangée de la patte numéro trois du capteur et la broche 5 V du connecteur en bas à gauche de la carte.
5. Vérifie les branchements en relisant la procédure. Nous pouvons maintenant passer à la rédaction du texte source.

Rédaction du texte source de

BagMagik1

Dans la première version du projet, nous allons seulement vérifier que la télécommande renvoie des codes et que nous réussissons à les décoder.

Pour faciliter ce décodage, nous allons utiliser des fonctions qui se trouvent dans une librairie créée spécialement pour exploiter un capteur d'ondes infrarouges tel que le TSOP que nous venons d'implanter. Il faudra installer cette librairie. Pour ce faire, nous devrons nous familiariser avec la programmation objets.

Commence par saisir le texte source suivant sans t'inquiéter des lignes qui contiennent des techniques encore inconnues. Il y a en effet plusieurs nouveautés importantes que je présente après le Listing 8.1.

Listing 8.1 : Version 1 de BagMagik

```
// BagMagik1
// OEN170314

// Ici viendra une ligne

const int    broCapteurIR = 10;      //
Broche 10 pour le capteur
IRrecv         oCapIR(broCapteurIR);
decode_results oCodeIR;

void setup()
{
  Serial.begin(9600);
  oCapIR.enableIRIn();      // Démarre
récepteur IR
  Serial.println("Appuie sur une touche de
```

```
la telco (V 1)...");
}

void loop() {
  if (oCapIR.decode(&oCodeIR)) {
  //
     Serial.println(oCodeIR.value, DEC);
     Serial.println();
    digitalWrite(13, HIGH);
    oCapIR.resume();              // Code
suivant
     delay(250);
  }
}
```

La première version de ce nouveau projet n'est pas beaucoup plus longue que les projets précédents, mais elle présente une nouveauté immense : il y a des objets. Alors, qu'est-ce qu'un objet ?

Des objets pour programmer !

Jusqu'à maintenant, tu as utilisé :

- des fonctions, qui consomment du temps pour réaliser des actions ;
- des variables, qui consomment de l'espace pour stocker des données en mémoire.

Dans le monde physique, il est souvent logique de regrouper des actions et des informations. Par exemple, un marteau est constitué d'un manche d'une certaine couleur et d'une tête d'une autre couleur. Ce sont les données du marteau, ses attributs. Le marteau a également la fonction `clouer()`, qui concentre une énergie

importante sur une surface dure pour, par exemple, enfoncer des clous.

De même, un animal possède des attributs et des actions. Il est d'une certaine couleur, il a un certain nombre de pattes, il sait manger, courir, crier, etc.

Les chercheurs en programmation ont assez vite compris l'intérêt de réunir des actions et des données dans un petit conteneur portant un nom. Imagine les trois variables et les trois fonctions suivantes :

```
int iCouleur;
int iPoids;
float fTaille;
void courir(vitesse);
void dormir(temps);
void aboyer(volume);
```

On peut réunir ces six définitions dans un joli coffret que l'on appelle une classe. Les éléments appartiennent à cette classe. Si je nomme la classe chien, je peux ensuite créer un objet à partir de cette classe, ce qui signifie instancier la classe.

Pour créer un objet à partir d'une classe, j'utilise le nom de la classe comme si c'était un nouveau type. Voici l'écriture générique :

```
nomClasse  nomNouvelObjet();
```

Et voici un exemple concret de création d'objet :

```
chien  monToutou();
```

Je peux ensuite lire et écrire les données qui appartiennent à cet objet comme ceci :

```
monToutou.iPoids = 12;
```

Je peux déclencher une action de l'objet ainsi :

```
monToutou.aboyer(3);
```

La variable et la fonction, pardon la méthode, appartiennent à l'objet. Pour les utiliser, il est obligatoire d'ajouter le « nom de famille » , c'est-à-dire le nom de l'objet propriétaire, avec un point séparateur.

Les classes et les objets offrent de nombreux avantages que tu découvriras en temps utile.

Premier essai de compilation

Après avoir saisi le texte source de ce projet (ou l'avoir récupéré dans les exemples, mais c'est en codant qu'on devient coderon), je te propose de lancer une vérification. Inutile de brancher la carte. Nous ne sommes pas encore prêts à téléverser le programme dans le mikon.

1. Lance une compilation du texte source par la commande **Vérifier**.

   Normalement, cela ne doit pas bien se passer. Tu dois voir apparaître plusieurs lignes de messages en orange dans le panneau inférieur de l'atelier (Figure 8.5).

Cette erreur est tout à fait normale. Nous citons dans le texte source un identifiant, qui est celui de l'objet `IRrecv`, mais le compilateur ne trouve aucune définition correspondante, ni dans le texte source, ni dans aucune des librairies prédéfinies installées d'office avec l'atelier. La librairie qu'il faut ajouter à notre atelier doit être récupérée sur le Web puis mise en place au bon endroit.

Figure 8.5 : Compilation impossible par absence de librairie.

Installons une librairie

Plusieurs librairies sont disponibles pour gérer avec plus de confort un capteur infrarouge tel que celui implanté dans la carte d'essai. Parmi celles-ci, il y en a une qui remplit très bien son office : celle de Ken Shirriff. Elle est bien sûr gratuite et disponible avec son texte source.

Le nom que doit porter le sous-dossier dans lequel la librairie sera rendue disponible pour l'atelier devra s'écrire exactement `IRremote`. Lorsque tu télécharges la librairie sur le Web, elle porte un nom plus long, qu'il faudra donc modifier.

1. Avec ton navigateur Web, accède à la page d'accueil du gisement correspondant à la librairie IRremote, sur le site github. Tu peux faire une recherche de github IRremote pour éviter de saisir l'adresse :

`https://github.com/z3t0/Arduino-IRremote`

2. Tu vois apparaître une liste de noms de fichiers. Sur la droite se trouve un bouton vert que tu cliques pour choisir l'option **Download ZIP**.

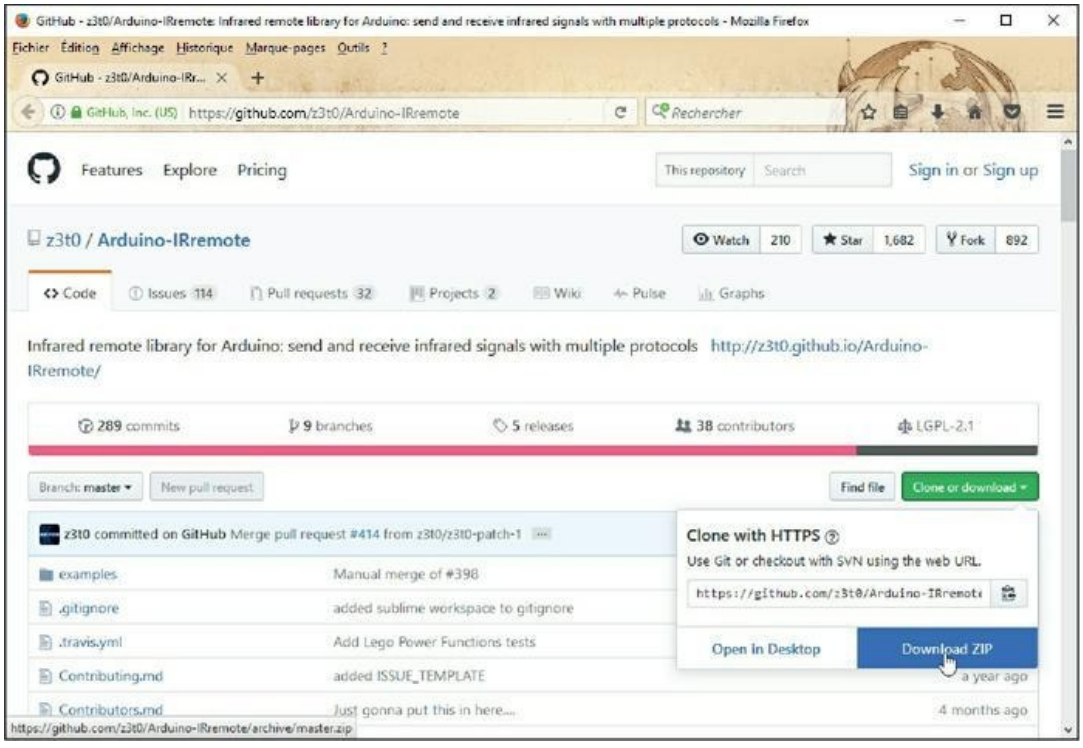


Figure 8.6 : Page d'accueil du gisement de la librairie.

3. Une boîte standard est apparue pour te demander ce que tu veux faire du fichier archive. Réponds que tu veux enregistrer le fichier. Tu remarques que le fichier porte un nom différent de IRremote.

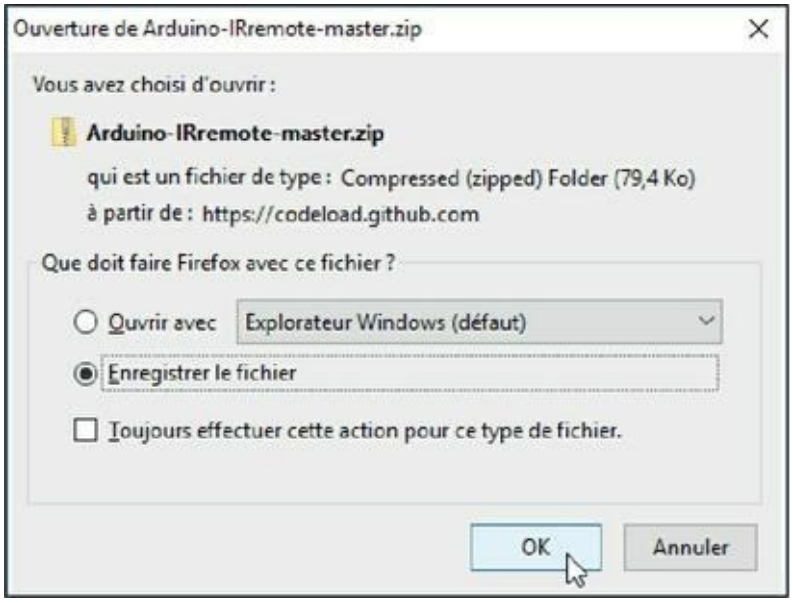

Ouverture de Arduino-IRremote-master.zip
Vous avez choisi d'ouvrir :
Arduino-IRremote-master.zip
qui est un fichier de type : Compressed (zipped) Folder (79,4 Ko)
à partir de : https://codeload.github.com
Que doit faire Firefox avec ce fichier ?
Ouvrir avec
Explorateur Windows (défaut)
Enregistrer le fichier
Toujours effectuer cette action pour ce type de fichier.
OK
Annuler

Figure 8.7 : Boîte standard de téléchargement.

4. Une autre boîte te demande de décider du lieu de stockage de l'archive. Choisis par exemple le dossier `Téléchargements`.

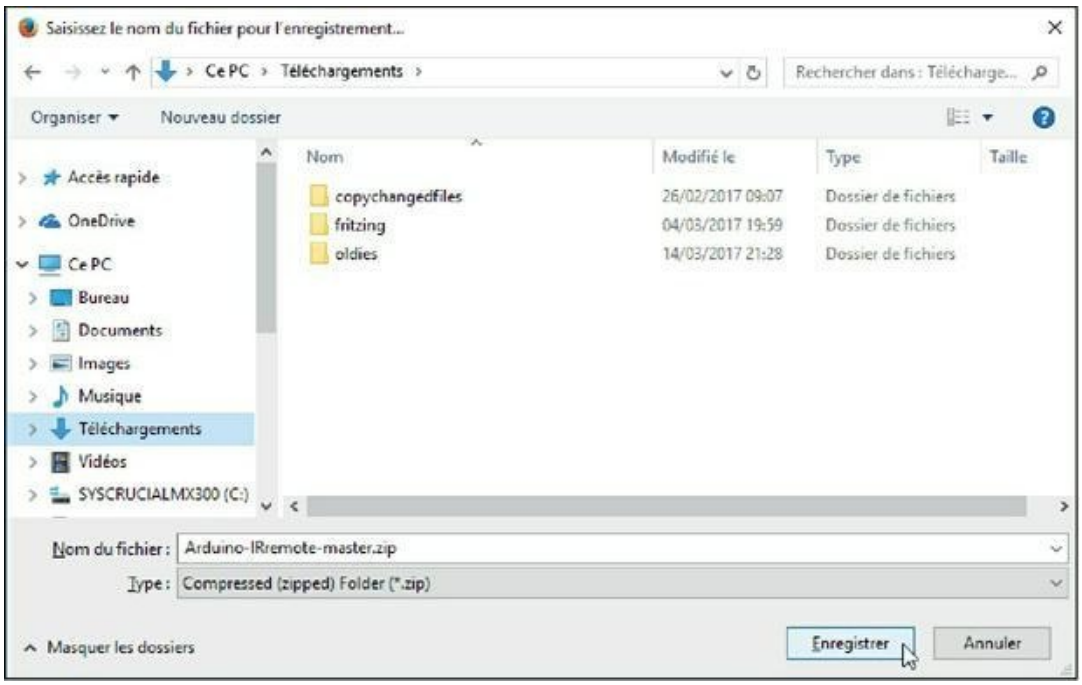


Figure 8.8 : Choix du lieu de stockage de l'archive.

Le dossier de la librairie est maintenant prêt à être réimplanté.

5. Démarre si nécessaire l'atelier Arduino. Dans le menu **Croquis**, choisis **Inclure une bibliothèque** puis sélectionne la commande **Ajouter la bibliothèque .ZIP**.

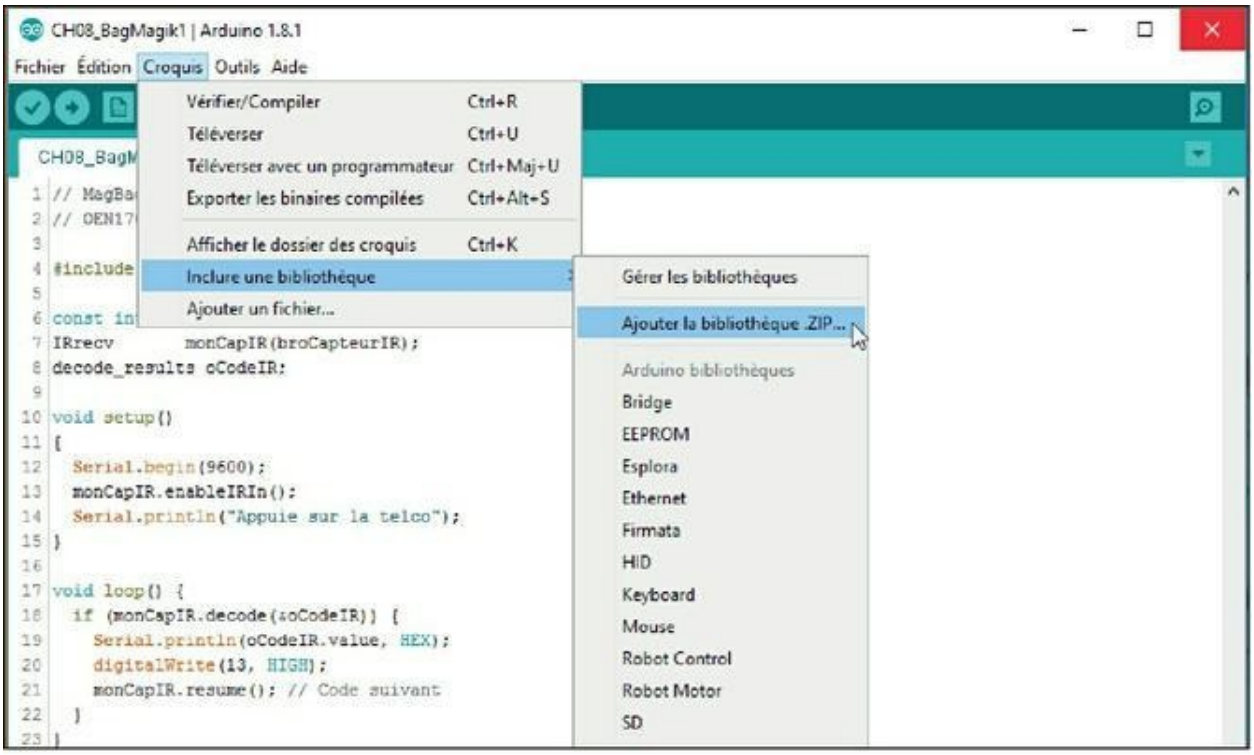


Figure 8.9 : Commande de chargement d'une librairie au format archive.

Une boîte de choix de fichier apparaît (Figure 8.10).

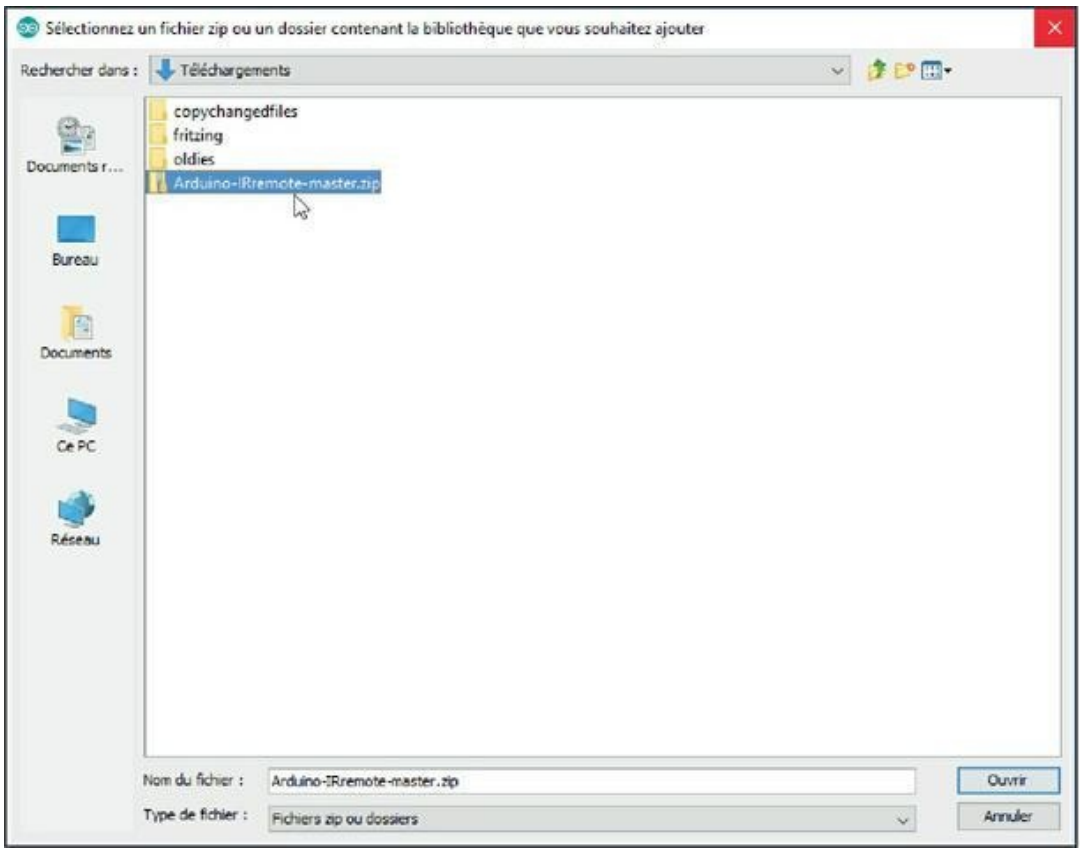


Figure 8.10 : Choix du fichier archive de librairie.

6. Navigue jusqu'à ton dossier `Téléchargements` et sélectionne le fichier, puis confirme avec **Ouvrir** (ou double-clique le nom directement).

   Il n'y a plus qu'à patienter un petit peu.

7. Quitte l'atelier Arduino.

La librairie est maintenant disponible, mais pour plus de sûreté, nous allons modifier son nom.

Renommons la librairie

La librairie a été implantée dans `Mes documents, Arduino`.

1. Ouvre ton navigateur, Finder ou Gestionnaire de fichiers, et affiche le contenu du dossier suivant (soit `Documents`, soit `Mes documents`) :

   ```
   Documents, Arduino, libraries
   ```

   Tu dois voir au moins un dossier, celui de la librairie qui vient d'être installée :

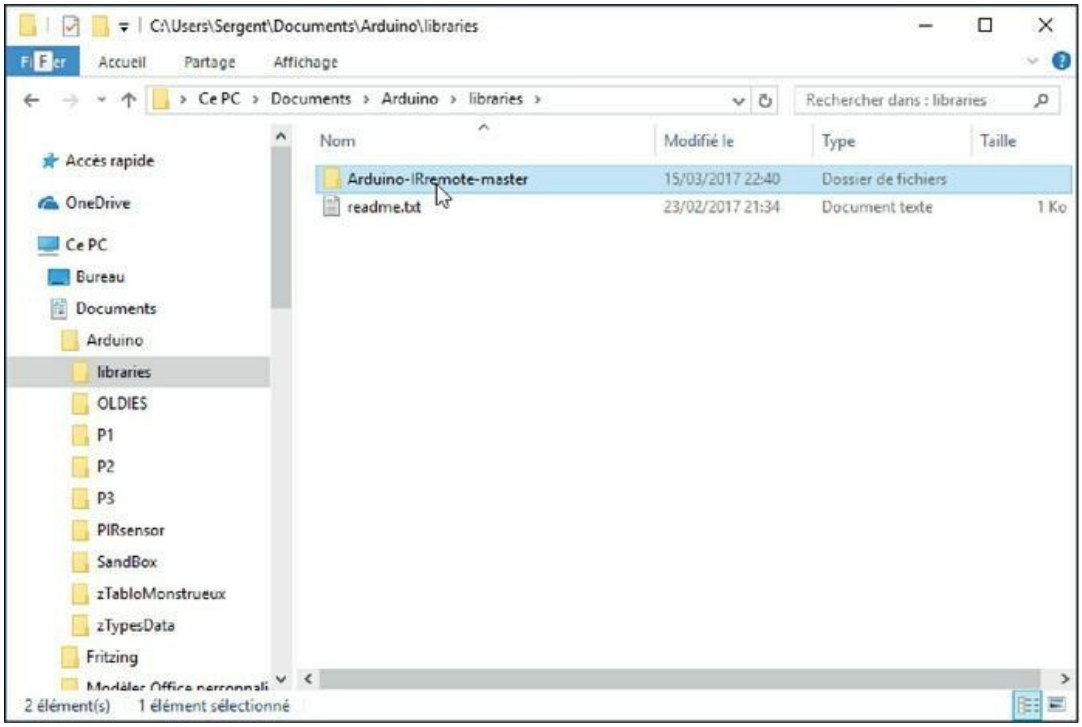


Figure 8.11 : Accès au dossier de la librairie.

2. Sélectionne le nom du dossier (chez moi, il s'écrit `Arduino-IRremote-master`) et utilise la commande de ton système pour changer le nom (c'est en général **Renommer** ou par simple clic dans le nom).

   Voici le nouveau (et véritable) nom de la librairie :

   ```
   IRremote
   ```

Attention aux majuscules et minuscules !

3. Démarre l'atelier Arduino. Normalement, le dernier projet en cours est rechargé, donc `BagMagik1` (recharge-le si nécessaire).

4. Lance une vérification. Dorénavant, elle doit réussir, malgré quelques messages en orange.

CH08_BagMagik1 | Arduino 1.8.1
Fichier Édition Croquis Outils Aide
CH08_BagMagik1
// MagBag1
// OEN170314
#include <IRremote.h>
const int broCapteurIR = 10; // Broche num 10 pour le capteur
IRrecv monCapIR(broCapteurIR);
decode_results oCodeIR;
void setup()
{
Serial.begin(9600);
monCapIR.enableIRIn();
Serial.println("Appuie sur la telco");
}
void loop() {
if (monCapIR.decode(&oCodeIR)) {
Serial.println(oCodeIR.value, HEX);
digitalWrite(13, HIGH);
monCapIR.resume(); // Code suivant
Compilation terminée.
In file included from C:\Program Files (x86)\Arduino\libraries\IRremote\ir_Lego_PF.cpp:3:0:
C:\Program Files (x86)\Arduino\libraries\IRremote\ir_Lego_PF_BitStreamEncoder.h: In member function
C:\Program Files (x86)\Arduino\libraries\IRremote\ir_Lego_PF_BitStreamEncoder.h:107:38: warning: int
return STOP_PAUSE_DURATION + 5 * MAX_MESSAGE_LENGTH - messageLength;
Le croquis utilise 7528 octets (23%) de l'espace de stockage de programmes. Le maximum est de 32256
Les variables globales utilisent 433 octets (21%) de mémoire dynamique, ce qui laisse 1615 octets po
Arduino/Genuino Uno sur COM3

Figure 8.12 : Compilation réussie, avec des messages.

Compilation : erreurs ou avertissements ?

Jusqu'à présent, lorsque tu voyais un message orange, c'était mauvais signe : le texte source présentait une erreur empêchant de compiler. Mais tous les messages ne sont pas aussi graves. Il existe des avertissements (warnings, en anglais) que tu peux consulter ou ignorer, car ils n'empêchent pas de générer le fichier binaire à téléverser.

Un réglage de la sévérité du compilateur est disponible dans les préférences. Au départ, il est réglé à fond : tous les messages sont affichés. Tu peux diminuer la sévérité d'un cran pour afficher moins d'avertissements :

1. Dans l'atelier Arduino, ouvre le menu **Fichier** (Windows et Linux) ou le menu **Arduino** (Mac OS) pour choisir **Préférences** et ouvrir la boîte des options.

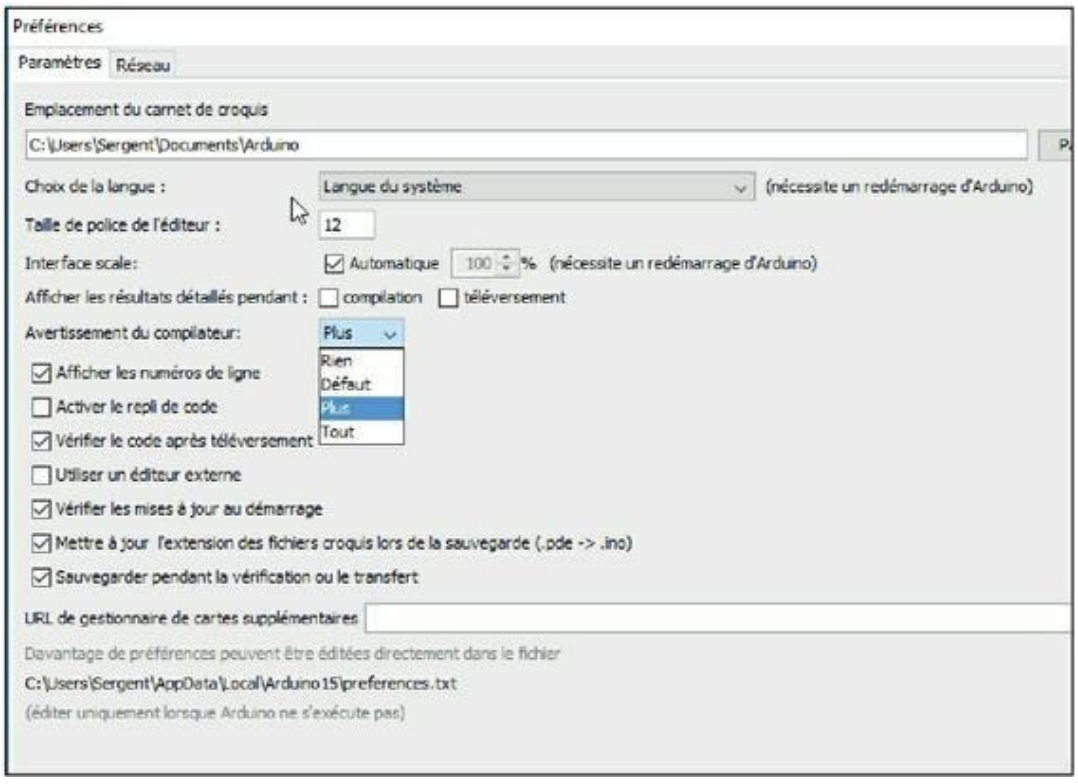


Figure 8.13 : Les préférences de l'atelier Arduino.

2. Ouvre la liste de l'option **Avertissements du compilateur** et sélectionne **Plus** au lieu de **Tout**. Referme la boîte en validant par OK.

Nous allons (enfin) tester le programme puis revenir sur chacune des nouvelles techniques qu'il contient.

Test de BagMagik1

Si tu es arrivé à cette section, c'est que le programme se compile. Les deux messages « Le croquis utilise… » et « Les variables globales… » sont apparus.

1. Rebranche la carte et attends 2 à 3 secondes.
2. Lance le téléversement. N'hésite pas à adopter le raccourci clavier correspondant à la commande Téléverser : c'est **Ctrl** + **U**.
3. Ouvre la fenêtre du Moniteur série et attends le premier message.
4. Dès que le message d'accueil apparaît, prends ta télécommande et vise le capteur infrarouge. Appuie sur l'une des touches les plus faciles à trouver (nous allons les détecter dans la version suivante du programme).

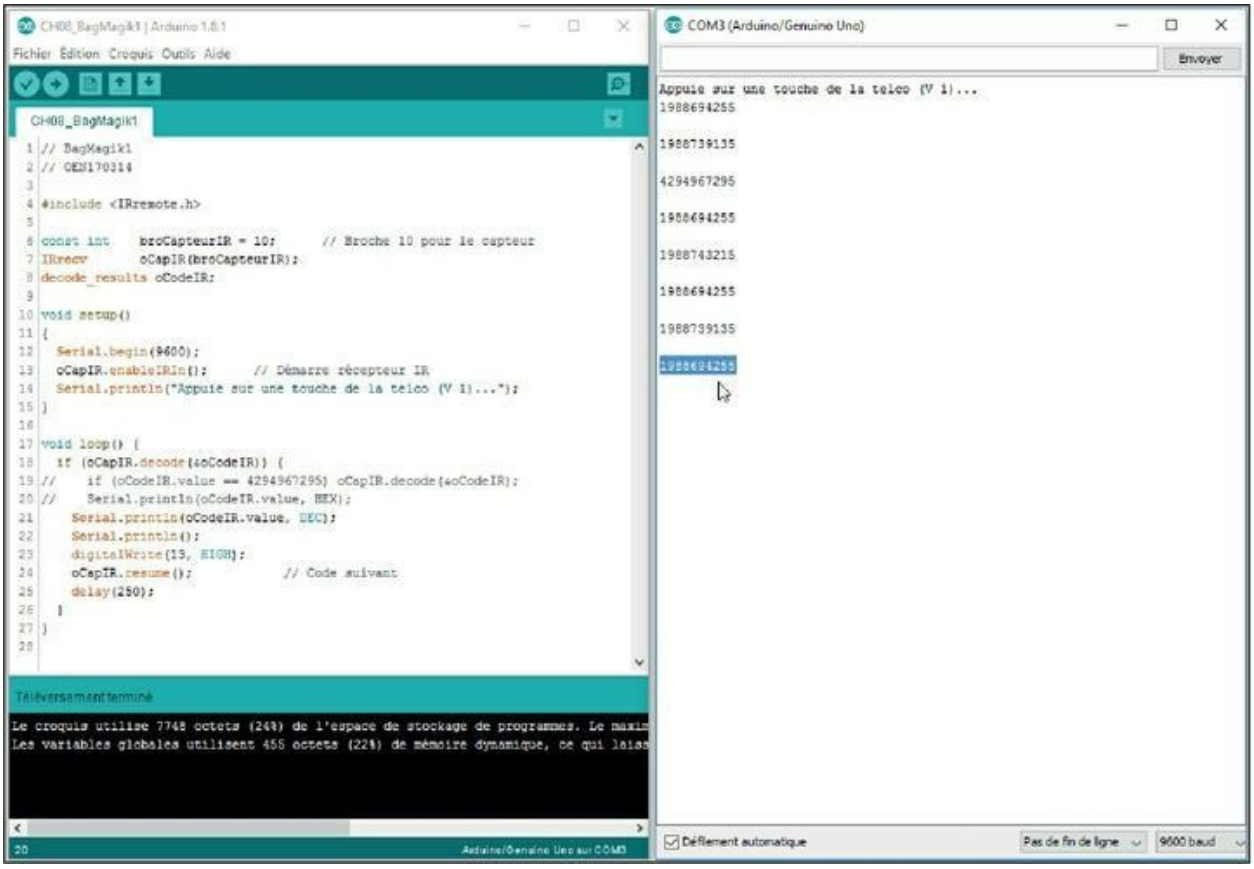


Figure 8.14 : Exécution de BagMagik1.

Si tu vois des codes ayant la valeur 4294967295, ignore-les. Ce sont des codes de répétition parce que la touche a été appuyée un peu trop longtemps.

Voici par exemple le code de la touche LECT de ma télécommande de type NEC :

1988694255

Bien sûr, ta télécommande va envoyer d'autres valeurs pour les mêmes touches.

5. Prends de quoi noter les valeurs numériques des touches que tu veux utiliser. Il te faut choisir trois

touches. J'ai par exemple choisi les touches LECT, PAUSE et AV. RAP. d'une télécommande qui appartenait à un vieux magnétoscope.

Conserve bien tes codes, car nous allons en avoir besoin dans la prochaine version du projet. Mais d'abord, partons à la découverte du texte source avec une première question.

Comment le compilateur sait-il où trouver les définitions de classes pour créer des objets dans le programme ? En lui indiquant où aller les chercher, pardi !

En cas de conflit de librairies…

Normalement, le simple fait de changer le nom du dossier de la librairie en `IRremote` suffit à régler un éventuel problème de conflit avec une autre librairie. Si l'erreur de compilation est toujours là, il faut supprimer cette autre librairie, quitte à la réinstaller plus tard quand tu en auras besoin. Si tu rencontres ce problème, sers-toi des explications que j'ai renvoyées tout à la fin de ce chapitre.

Une directive d'inclusion

Ton programme ne peut pas utiliser les nouvelles fonctions de ce projet simplement parce que tu cites leur nom dans ton texte source. Lorsque tu écris un appel à une fonction, le compilateur trouve les instructions de la fonction (car il a besoin de les compiler) seulement dans les deux cas suivants :

» C'est une fonction prédéfinie, installée en même temps que l'atelier IDE. C'est le cas de toutes les

fonctions vues jusqu'à présent : `pinMode()`, `digitalRead()`, *etc.*

» C'est une fonction locale, c'est-à-dire une fonction que tu as créée, et dont le corps avec toutes les instructions se trouve un peu plus bas dans le même texte source. Nous avons déjà créé au moins une fonction « personnelle » dont nous pouvions choisir le nom.

En dehors de ces deux cas, tu dois informer le compilateur que tu vas utiliser une fonction qui se trouve dans une librairie complémentaire.

Pour réaliser cette déclaration, il faut ajouter tout au début du programme ce que l'on appelle une directive. Les directives commencent toujours par un signe # (et ne se terminent jamais par un signe point-virgule !).

Ici, il nous faut la directive #include que l'on fait suivre par le nom de la librairie. Voici comment l'utiliser :

```
#include  <nomFichierDeLibrairie>
```

La directive est suivie du nom d'un fichier dit d'en-tête (fichier `.h` comme Header). Ce fichier doit se trouver dans le sous-dossier de la librairie que nous avons installée. Voici la directive qui est déjà présente dans notre texte source (voir le Listing 8.1) :

```
#include <IRremote.h>
```

Lorsque le nom du fichier est entouré d'un signe plus petit < à gauche et d'un signe plus grand > à droite, cela signifie que l'atelier doit chercher ce fichier dans tous les dossiers standard des librairies, c'est-à-dire dans le sous-dossier `libraries` de là où

est installé le programme. Dans notre cas, il s'agit du sous-dossier `libraries` de `Mes documents, Arduino`.

En revanche, lorsque le nom du fichier à inclure est entouré de guillemets, le fichier est d'abord cherché dans le dossier du texte source de ton projet.

La compilation réussit parce que deux conditions sont remplies :

- La librairie `IRremote` est installée.
- Une directive indique qu'il faut aller fouiller dans cette librairie pour trouver ce qui n'est pas défini dans le texte source lui-même.

Voyons donc à quoi nous sert cette librairie. Revoici le code source dans l'éditeur, afin de se repérer facilement grâce aux numéros de lignes en marge gauche (Figure 8.15).

```
1  // BagMagik1
2  // OEN170314
3
4  #include <IRremote.h>
5
6  const int     broCapteurIR = 10;      // Broche 10 pour le capteur
7  IRrecv        oCapIR(broCapteurIR);
8  decode_results oCodeIR;
9
10 void setup()
11 {
12   Serial.begin(9600);
13   oCapIR.enableIRIn();      // Démarre récepteur IR
14   Serial.println("Appuie sur une touche de la telco (V 1)...");
15 }
16
17 void loop() {
18   if (oCapIR.decode(&oCodeIR)) {
19     //
20     Serial.println(oCodeIR.value, DEC);
21     Serial.println();
22     digitalWrite(13, HIGH);
23     oCapIR.resume();            // Code suivant
24     delay(250);
25   }
26 }
27
```

Figure 8.15 : Édition de BagMagik1.

Comment s'utilise un objet ?

Observe la deuxième ligne dans les déclarations globales en début de fichier (numéro de ligne 7) :

```
IRrecv        oCapIR(broCapteurIR);
```

Cela ressemble à une déclaration de variable, sauf que le type est un drôle de type. Ce n'est ni int, ni boolean, ni long, ni void, etc. C'est parce que c'est un nom de classe. La librairie que nous avons installée, `IRremote`, contient la définition de cette classe. La ligne ci-dessus sert à créer un objet qui va incarner un récepteur d'ondes infrarouges.

Lors de la création de l'objet, nous devons fournir entre parenthèses le numéro de la broche sur laquelle doit travailler l'objet pour interroger le capteur infrarouge.

Cette déclaration crée en un geste l'objet et toutes les variables qu'il possède (les données membres).

Dans la ligne suivante, nous trouvons une autre création d'objet :

```
decode_results oCodeIR;
```

Je suis allé vérifier parmi les nombreux fichiers que regroupe la librairie `IRremote` : ce second objet est beaucoup plus simple que le précédent. Il ne définit aucune fonction, seulement des variables (des données membres).

J'ai pris soin de faire commencer tous les noms d'objets par la lettre minuscule `o`, pour que tu saches à tout moment qu'il s'agit d'un objet, et non d'une variable ou d'une fonction.

Préparation de l'objet

Dans la fonction de configuration `setup()`, il n'y a qu'une ligne remarquable, la deuxième :

```
oCapIR.enableIRIn();
```

Cette instruction est un appel à une fonction de l'objet (une méthode). Cette méthode fait démarrer le récepteur infrarouge en sorte qu'il se tienne prêt à détecter des codes. La fonction porte le nom `enableIRIn()`, (« activer entrée infrarouge »), mais comme elle appartient à un objet, nous devons préciser le nom de l'objet propriétaire de la fonction en séparant les deux par un point.

Une fonction qui appartient à un objet est en général désignée sous le nom de

méthode (ou parfois de fonction membre).

Utilisation de l'objet

Nous arriverons ensuite directement dans la fonction de boucle `loop()`.

Cette fonction principale est entièrement occupée par un bloc conditionnel if. Autrement dit, lorsque l'expression de test est fausse, on passe directement au tour suivant sans rien faire, pas même la pause d'un quart de seconde.

Étudions cette expression conditionnelle (ligne 18) :

```
if (oCapIR.decode(&oCodeIR)) {
```

L'expression à tester est en fait un appel à une fonction, pardon une méthode de notre objet `oCapIR` ! Cette méthode `decode()` renvoie une valeur différente de 0 si un code de touche a été détecté.

Ce qui est intrigant est le petit caractère & (on dit « ET commercial ») qui est accolé au début du nom de l'objet `oCodeIR`. Ce simple caractère a une importance énorme dans les langages de programmation de type C tel que celui de l'atelier Arduino.

Normalement, lorsque tu indiques le nom d'une variable dans les parenthèses d'un appel de fonction, c'est la valeur que possède la variable à ce moment qui est transmise à la fonction pour qu'elle travaille avec.

Mais ici, nous transmettons à la fonction `decode()` un objet. Cet objet n'a pas de valeur au sens habituel : il contient des variables qui possèdent chacune une valeur. Nuance. Grâce au symbole `&`, c'est l'adresse à laquelle est stocké l'objet dans la mémoire qui est transmise à la fonction. Celle-ci peut ainsi remplir les variables de l'objet, ce qui nous permettra d'y accéder ensuite pour en obtenir la valeur.

Ouf ! C'est peut-être l'heure de faire une petite pause, non ? N'hésite pas à relire le paragraphe précédent. Comprendre ce mécanisme va beaucoup te faire progresser. Et dis-toi que si un petit carré de sable peut en faire son affaire, tu dois pouvoir y parvenir aussi.

La méthode `decode()` a un double effet :

- Elle renvoie une valeur différente de 0 seulement si elle ne revient pas bredouille.
- Elle remplit un élément de données de l'objet `oCapIR` avec une valeur qui correspond à la touche enfoncée sur la télécommande.

Pour certains modèles de télécommandes qui envoient une valeur spéciale en cas de répétition de touche, il peut être utile d'ajouter un autre test afin d'ignorer ce code de répétition. Saurais-tu écrire cette instruction ? (Réponse en fin de chapitre.)

La ligne suivante demande de lire la valeur de la donnée nommée `value` qui appartient à notre objet `oCapIR`. Tu constates que ce n'est pas toi qui as choisi le nom de la donnée. Ce nom est défini par la classe dont est issu notre objet. Voilà pourquoi le nom complet est moitié anglais, moitié français. Nous faisons afficher la

valeur au format décimal par l'option DEC. (Nous pourrions l'afficher en hexadécimal avec l'option HEX.)

Suit une instruction facile pour allumer la LED 13 soudée sur la carte, ce qui permet de confirmer quand un appui sur une touche a été détecté.

Nous arrivons à l'avant-dernière ligne :

```
oCapIR.resume();
```

Il s'agit d'un appel à une autre méthode de notre objet. La méthode `resume()` sert à reconfigurer l'objet pour qu'il soit prêt à capter le prochain code de touche.

Après ces longues explications, je pense que le programme t'est devenu moins étrange. Nous créons des objets et nous utilisons leurs méthodes et leurs données grâce à la notation fondée sur le point entre nom d'objet et nom d'élément d'objet.

La version 2 de BagMagik

La première version du projet nous a permis de vérifier que le capteur fonctionnait et détectait bien les codes des touches de la télécommande. Nous pouvons maintenant envisager de faire contrôler la logique du programme par les touches frappées. Nous allons en profiter pour découvrir une nouvelle technique d'exécution conditionnelle, le bloc switch..case.

La gare de triage switch

Tu possèdes sans doute un telliphone (les anglophones disent « smartphone ») ? Tu as donc sans doute interrogé et personnalisé ta messagerie vocale. Tu connais donc le principe des menus vocaux :

« Pour écouter vos messages, tapez 1. Pour modifier votre annonce, tapez 2... »

Pour programmer les réponses à ces questions, tu peux aligner toute une série de tests dans ce style :

```
if (touche == 1) lireMsg();
if (touche == 2) modifierAnnonce();
//...
if (touche == 9) toutEffacer();
```

Lorsque la valeur à tester reste la même dans plusieurs cas, le mot clé switch permet d'éviter la multiplication de tests fondés sur if :

```
switch (touche) {
  case 1:
    lireMsg();
     break;
  case 2:
    modifierAnnonce();
     break;

  //...
  case 9:
    toutEffacer();
     break;
  default:
   // si touche inconnue ?
}
```

Le switch est délicat

Plusieurs remarques :

1. Il s'agit bien d'un bloc : il y a une accolade sur la ligne de tête et une autre tout en bas pour fermer le bloc switch.

2. Le nombre de cas n'est pas limité, mais la valeur à comparer doit être figée, c'est-à-dire être une constante. Soit tu indiques directement une valeur comme nous le faisons ici, soit tu déclares une constante en début de programme :

   ```
   const long LECTURE = 1988694255 ;
   ```

   Tu peux dans ce cas écrire :

   ```
   case LECTURE :
   ```

   En revanche, tu ne peux pas indiquer un nom de variable après case.

3. Tous les cas doivent se terminer par l'instruction `break ;` pour ignorer tous les cas suivants jusqu'à l'accolade fermante du bloc entier. Si tu l'oublies, les cas suivants seront exécutés, même si la valeur ne convient pas. C'est un vrai piège, mais c'est parfois utile.

Nous pouvons maintenant voir comment ce bloc switch est utilisé dans le projet. Il s'étend sur presque 20 lignes !

Listing 8.2 : Version 2 de BagMagik

```
// BagMagik2
// OEN170315
```

```
#include <IRremote.h>

const int    broCapteurIR = 10;      //
Broche 10 pour le capteur
IRrecv       oCapIR(broCapteurIR);
decode_results oCodeIR;
unsigned long gTouche;              // AJOUT2
const long REPET = 4294967295;    // AJOUT2

void setup()
{
  Serial.begin(9600);
  oCapIR.enableIRIn();      // Démarre
récepteur IR
  Serial.println("Appuie sur une touche de
la telco (V 2)...");
}

void loop() {
 if (oCapIR.decode(&oCodeIR)) {
    gTouche = oCodeIR.value;
    digitalWrite(13, HIGH);
    switch (gTouche) {
      case 1988694255:          // Touche
LECT
        Serial.println(gTouche, DEC);
        Serial.println("* Touche 1");
         break;
      case 1988739135:          // Touche
AV.RAP
        Serial.println(gTouche, DEC);
        Serial.println("** Touche 2");
```

```
          break;
       case 1988743215 :          // Touche
AV.RAP
         Serial.println(gTouche, DEC);
         Serial.println("*** Touche 3");
          break;
       case  REPET:           // Code de
repetition
          break;
        default:
          break;
     }
     Serial.println();
     oCapIR.resume();           // Code
suivant
  }
  gTouche = 0;
  delay(250);
}
```

Dans la première ligne du bloc switch, nous utilisons comme valeur à tester une variable déclarée en début de programme du type long, c'est-à-dire une variable deux fois plus spacieuse qu'un int. Cette variable `gTouche` contient la valeur qui a été extraite de l'objet `oCodeIR` quelques lignes plus haut.

Le deuxième sous-bloc case indique comme valeur le code décimal de la touche de lecture. Bien sûr, tu dois remplacer cette valeur par le code de la touche que tu as choisie sur ta télécommande.

Lorsque la valeur de `gTouche` est identique à cette valeur littérale, nous exécutons les trois instructions qui suivent. Les deux premières servent à afficher la valeur ainsi qu'un petit message, et

la troisième est la fameuse instruction spéciale break; qu'il ne faut jamais oublier.

Les deux sous-blocs suivants permettent de capturer d'autres touches. Pour finir, je prévois un fourre-tout qui s'appelle default : . Tu peux y placer des instructions spéciales, comme faire afficher un message pour dire que la touche frappée n'est aucune de celles attendues.

Tu remarques que je remets la variable gTouche à 0 à la fin du traitement.

Les autres instructions existaient déjà dans la première version.

Tu peux maintenant lancer un test de cette version 2. Voici ce que tu devrais obtenir après avoir utilisé la télécommande (Figure 8.16).

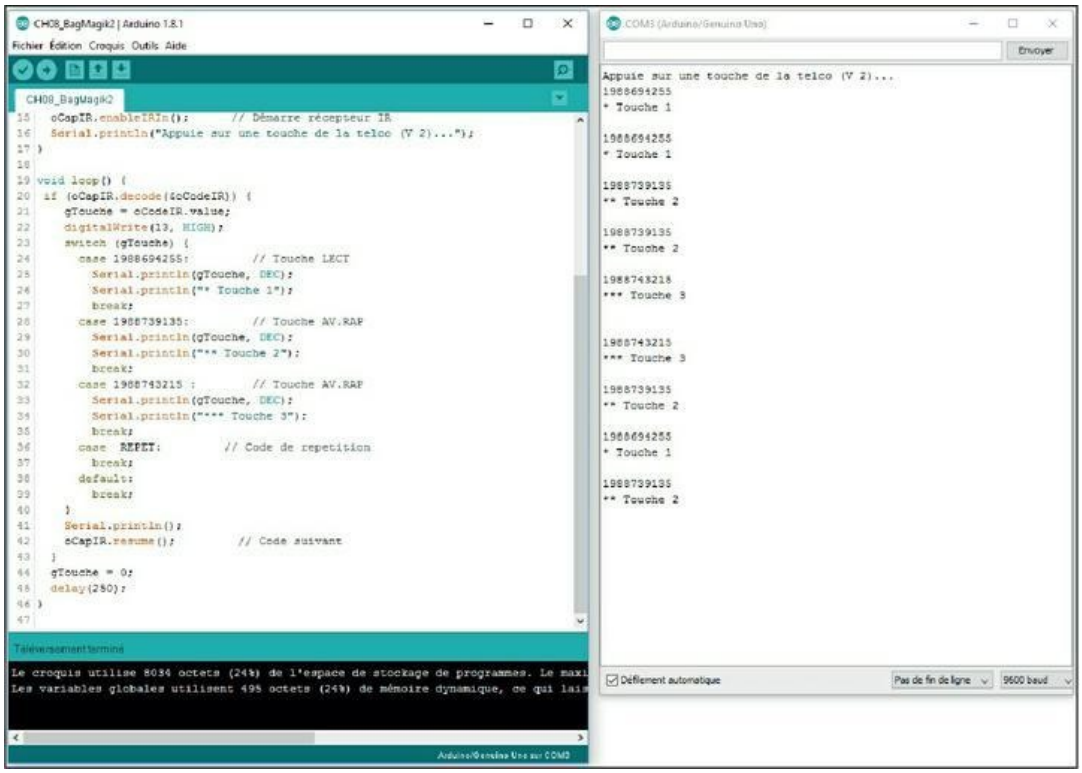


Figure 8.16 : Exécution de la version 2 de BagMagik.

Vérifie que le programme reconnaît bien les trois touches que tu as choisies.

Pour la version finale de notre projet, je te propose de découvrir un autre concept essentiel de programmation : le tableau de valeurs.

La version 3 de BagMagik

Nous allons nous servir de notre télécommande pour puiser dans une sorte de petit dictionnaire contenant des sujets, des verbes, des adverbes et des compléments. Mais comment faire pour automatiser la lecture d'une valeur parmi plusieurs ?

Nous décidons de puiser dans le stock de mots suivant :

- trois sujets ;
- trois verbes ;
- trois adverbes ;
- trois compléments.

Nous ferons quatre tours de boucle pour faire choisir avec la télécommande d'abord le sujet, puis le verbe, etc.

Il va falloir déclarer douze variables pour les douze mots. La logique va ressembler à ceci (en pseudocode) :

```
Si touche1 ET Tour1 Afficher Sujet1
Si touche2 ET Tour1 Afficher Sujet2
Si touche3 ET Tour1 Afficher Sujet3
Si touche1 ET Tour2 Afficher Verbe1
Si touche2 ET Tour2 Afficher Verbe2
Si touche3 ET Tour2 Afficher Verbe3
Si touche1 ET Tour3 Afficher Adverbe1
Si touche2 ET Tour3 Afficher Adverbe2
Si touche3 ET Tour3 Afficher Adverbe3
Si touche1 ET Tour4 Afficher Complément1
Si touche2 ET Tour4 Afficher Complément2
```

```
Si touche3 ET Tour4 Afficher Complément3
```

Ouf ! Comment faire pour ne pas avoir toutes ces variables ?

La solution est le tableau de données. Nous en avons déjà utilisé un pour créer le jeu de lecture binaire `Devinum`. Mais c'était un tableau linéaire, comme un train, un tableau à une seule dimension :

```
byte ayTablo[10];   // Projet Devinum
```

Pourquoi ne pas entrer dans la seconde dimension ? Allez, on saute !

Si un tableau possède deux dimensions, on peut imaginer des lignes et des colonnes, bref, comme un vrai tableau, ou comme les pixels d'un écran qui sont aussi en lignes et en colonnes.

Il nous faut quatre lignes et trois colonnes. Mais de quoi ? Quel type de données ? Le type chaîne de caractères, c'est-à-dire plusieurs caractères à la queue leu leu.

Pour un seul caractère, le type est `char`, comme dans :

```
char maLettre = 'b';
```

Tu remarques que la lettre est délimitée par des apostrophes.

Pour déclarer une chaîne, ce sont des guillemets :

```
const char maPhrase = "Quand on descend en
ville.";
```

En réalité, une chaîne est un groupe de caractères, donc un tableau. Pour en stocker plusieurs dans un tableau, il faut une écriture spéciale pour le type :

```
const char* monTablo = {"Mot1", "Mot2",
"Mot3"};
```

Tu remarques :

- Le signe astérisque * accolé au nom du type char. Il signifie que nous travaillons avec l'adresse mémoire du début de chaque chaîne. Je n'en dis pas plus ici, car les pointeurs sont un sujet assez sophistiqué.
- Les chaînes sont séparées par des virgules et l'ensemble est emballé entre deux accolades.
- Les déclarations commencent par le mot const qui officialise le fait que ces mots sont statiques, définis dès le départ et ne peuvent pas changer pendant l'exécution du programme. Si tu n'indiques pas const, cela fonctionne mais il y aura des avertissements lors de la compilation.

Pour creuser le sujet des pointeurs (et d'autres), je te conseille mon autre livre chez le même éditeur, Programmer pour les Nuls.

Voici donc comment nous pouvons installer en mémoire un tableau de mots. Nous indiquons les deux dimensions entre deux paires de crochets après le nom du tableau. Les espaces en trop servent à rendre les lignes et colonnes plus lisibles.

```
const char* tabmots[4][3] = {
  {"Sujet1",    "Sujet2",    "Sujet3" },
  {" Verbe1",   " Verbe2",   " Verbe3" },
  {" Adverbe1", " Adverbe2", " Adverbe3" },
  {" Comple1.", " Comple2.", " Comple3." }
} ;
```

Pour puiser un mot dans le tableau et l'afficher, nous écrirons comme ceci :

```
Serial.print(tabmots[tour][0]);
```

En fonction de la valeur de `tour`, cela affichera Sujet1, Verbe1, Adverbe1 ou Comple1.

Nous allons ainsi pouvoir automatiser l'utilisation du tableau dans notre bloc conditionnel switch. Découvrons donc le texte source avant de passer au test.

Listing 8.3 : Version 3 de BagMagik

```
// BagMagik3
// OEN170315

#include <IRremote.h>

const int    broCapteurIR = 10;   // Broche
num 10 pour le capteur
IRrecv       oCapIR(broCapteurIR);
decode_results oCodeIR;
unsigned long  gTouche;
const long REPET = 4294967295;
byte mot = 0;                          //
AJOUT3

char* tabmots[4][3] = {              // AJOUT3
  {"Je", "Le chat", "La voisine" },
  {" mange", " pense", " bouge" },
  {" sans cesse", " lentement", " par pur
plaisir" },
```

```
  {" la nuit.", " le midi.", " depuis deux
jours." }
} ;

void setup()
{
  Serial.begin(9600);
  oCapIR.enableIRIn();      // Démarre
récepteur IR
  Serial.println("Appuie sur une touche de
la telco (V 3)...");
}

void loop() {
  if (oCapIR.decode(&oCodeIR)) {
    if (mot > 3) mot = 0;                 //
AJOUT3
    gTouche = oCodeIR.value;
   afficherCode(gTouche, mot);           //
AJOUT3
    digitalWrite(13, HIGH);
    oCapIR.resume();
    mot++;                                //
AJOUT3
  }
  if (mot > 3) { Serial.println(); mot = 0;
}  // AJOUT3
}
 // AJOUT3
void afficherCode(unsigned long code, byte
tour) {
  switch (code){
```

```
    case 1988694255 :         // Touche LECT
      Serial.print(tabmots[tour][0]);
       break;
    case 1988739135 :         // Touche
PAUSE
      Serial.print(tabmots[tour][1]);
       break;
    case 1988743215 :         // Touche
AV.RAP
      Serial.print(tabmots[tour][2]);
       break;
    // case REPET inutile, voir default:
    default:
       mot--;
       break;
    }
}
```

Saisis ce texte source en repartant de la précédente version, `BagMagik2`. J'indique les nouveautés par des commentaires de fins de lignes.

Analyse de BagMagik3

Nous déclarons une nouvelle variable pour avancer d'une ligne à la suivante dans le tableau, d'abord parmi les sujets, puis parmi les verbes, etc. Cette variable `mot` va varier entre 0 et 3, puis revenir à 0 :

```
byte mot = 0;
```

Arrive ensuite l'énorme déclaration du tableau de mots. Observe bien les virgules et les accolades internes. Voici le même tableau après ajout d'espaces d'alignement :

```
char* tabmots[4][3] = {

  {"Je",          "Le chat",    "La voisine"
},
  {" mange",      " pense",     " bouge"
},
  {" sans cesse", " lentement", " par pur
plaisir"    },
  {" la nuit.",   " le midi.",  " depuis deux
jours." }
} ;
```

Rien n'a changé dans la fonction de préparation `setup()`. La boucle principale a maigri : tout le bloc `switch` est venu former le contenu d'une nouvelle fonction qui est appelée depuis `loop()` seulement quand une touche est détectée.

Mais d'abord, nous testons la variable qui décide du numéro du tour de boucle. Si elle vaut plus de 3, c'est que nous avons fait quatre tours (de 0 à 3). Nous repartons donc au début d'une nouvelle phrase :

```
if (mot > 3) mot = 0;
```

Nous récupérons le code de touche dans une variable comme auparavant, puis nous appelons la nouvelle fonction avec ce code et le numéro de tour :

```
afficherUnMot(gTouche, mot);
```

Nous n'oublions pas d'incrémenter le compteur de tours :

```
mot++;
```

Une fois sortis du bloc conditionnel, nous pouvons tester si la phrase est terminée et, si oui, provoquer un saut de ligne tout en remettant le compteur à0:

```
if (mot > 3) { Serial.println(); mot = 0; }
```

Voyons les nouveautés de la fonction que nous appelons et, d'abord, sa ligne de tête :

```
void afficherUnMot(unsigned long code, byte
tour) {
```

Elle ne renvoie rien, mais attend deux paramètres d'entrée : le code de touche et le numéro de tour. Tu constates que le nom local des deux variables peut être différent du nom d'origine : `code` pour `gTouche` et `tour` pour `mot`. C'est une bonne pratique.

La suite de la fonction nous est déjà connue. La vraie nouveauté est l'accès à un mot du tableau, comme ceci :

```
Serial.print(tabmots[tour][0]);
```

Cette instruction lit le mot stocké à l'intersection entre :

- la première colonne du tableau (la colonne 0) ;
- la ligne désignée par `tour` qui peut valoir entre 0 et 3.

Une particularité concerne la branche fourre-tout default. Nous y faisons reculer le compteur de tours d'une position avec `mot--`. Pourquoi ? Parce que nous ne voulons pas passer à la ligne de mots suivante si la touche frappée n'est aucune de celles choisies pour le jeu.

Et maintenant, en avant pour les tests !

Test du projet

1. Téléverse le projet puis ouvre le Moniteur série.
2. Dès que le message de bienvenue apparaît, combine de différentes manières les trois touches choisies pour générer des phrases plus ou moins loufoques.

   Sais-tu combien de phrases différentes tu peux générer ?

Voici un aperçu des phrases possibles (Figure 8.17).

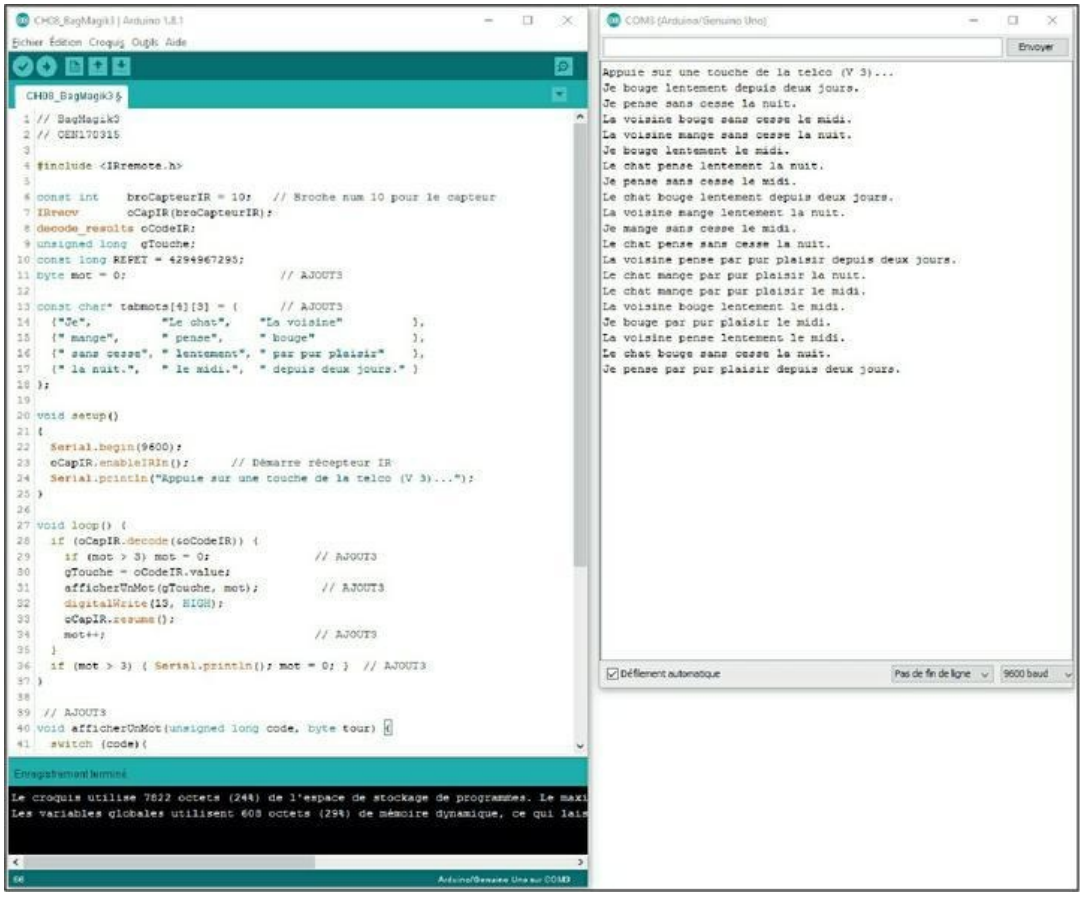

CH08_BagMagik3
// BagMagik3
// OEN170315
#include <IRremote.h>
const int broCapteurIR = 10; // Broche num 10 pour le capteur
IRrecv oCapIR(broCapteurIR);
decode_results oCodeIR;
unsigned long gTouche;
const long REPET = 4294967295;
byte mot = 0; // AJOUT3
const char* tabmots[4][3] = { // AJOUT3
{"Je", "Le chat", "La voisine" },
{" mange", " pense", " bouge" },
{" sans cesse", " lentement", " par pur plaisir" },
{" la nuit.", " le midi.", " depuis deux jours." }
};
void setup()
{
Serial.begin(9600);
oCapIR.enableIRIn(); // Démarre récepteur IR
Serial.println("Appuie sur une touche de la telco (V 3)...");
}
void loop() {
if (oCapIR.decode(&oCodeIR)) {
if (mot > 3) mot = 0; // AJOUT3
gTouche = oCodeIR.value;
afficherUnMot(gTouche, mot); // AJOUT3
digitalWrite(13, HIGH);
oCapIR.resume();
mot++; // AJOUT3
}
if (mot > 3) { Serial.println(); mot = 0; } // AJOUT3
}
// AJOUT3
void afficherUnMot(unsigned long code, byte tour) {
switch (code){
Enregistrement terminé
Le croquis utilise 7822 octets (24%) de l'espace de stockage de programmes. Le maxi
Les variables globales utilisent 608 octets (29%) de mémoire dynamique, ce qui lais
Envoyer
Appuie sur une touche de la telco (V 3)...
Je bouge lentement depuis deux jours.
Je pense sans cesse la nuit.
La voisine bouge sans cesse le midi.
La voisine mange sans cesse la nuit.
Je bouge lentement le midi.
Le chat pense lentement la nuit.
Je pense sans cesse le midi.
Le chat bouge lentement depuis deux jours.
La voisine mange lentement la nuit.
Je mange sans cesse le midi.
Le chat pense sans cesse la nuit.
La voisine pense par pur plaisir depuis deux jours.
Le chat mange par pur plaisir la nuit.
Le chat mange par pur plaisir le midi.
La voisine bouge lentement le midi.
Je bouge par pur plaisir le midi.
La voisine pense lentement le midi.
Le chat bouge sans cesse la nuit.
Je pense par pur plaisir depuis deux jours.
Défilement automatique
Pas de fin de ligne
9600 baud

Figure 8.17 : Quelques phrases produites avec BagMagik3.

Récapitulons

Et voici encore un chapitre copieux qui se termine ! Nous avons vu :

- l'exploitation d'un capteur infrarouge ;
- l'installation et l'utilisation d'une librairie de fonctions et de classes ;
- un premier aperçu de la programmation orientée objets ;
- les blocs conditionnels `switch case` ;
- les tableaux à deux dimensions et les chaînes.

Nous pouvons ainsi clore la Semaine 2. Dans la suivante, nous allons mettre en pratique, par des projets simples, toutes les techniques que nous venons de découvrir. À tout de suite (après un peu de repos !).

Réponse pour l'instruction de filtrage de code de répétition

Cette instruction est à insérer juste après le premier test conditionnel de BagMagik2, dans son bloc :

```
if (oCodeIR.value == 4294967295)
oCapIR.decode(&oCodeIR);
```

Chapitre 9
Que personne ne bouge !

AU MENU DE CE CHAPITRE :

- **Encore des infrarouges ?**
- **Le capteur PIR**
- **Faites du bruit !**
- **Ton système d'alarme personnelle**

La lumière infrarouge ? Oui, celle qui sert aux télécommandes. Mais elle ne sert pas qu'à changer de chaîne sur son téléviseur. Tout ce qui n'est pas à la même température que les objets qui l'entourent peut être détecté par son rayonnement infrarouge.

Bien sûr, la détection sera moins facile avec un animal à sang froid tel qu'un serpent. Je ne te garantis donc pas que le système d'alarme que nous allons construire dans ce chapitre va permettre de détecter l'arrivée de l'anaconda géant. En revanche, cela fonctionnera avec un être humain (pas un robot) et parfois même avec un petit animal de compagnie, si le réglage de sensibilité le permet.

Le capteur que nous allons utiliser est du type tout ou rien : soit il a détecté un mouvement, soit il n'a rien détecté. La sortie est donc du type numérique binaire : soit 0, soit 1.

Ce détecteur de proximité est en général désigné par les fabricants « capteur PIR » . Les deux lettres IR signifient « InfraRouge » . La lettre P a deux significations : Proximité ou Proche :

1. C’est un capteur de proximité, parce qu’il ne fonctionne qu’à quelques mètres au maximum de sa cible.
2. C’est un capteur conçu pour détecter les ondes infrarouges proches, dans une gamme de fréquences juste inférieure à celles des couleurs de lumière visibles. Il existe aussi des infrarouges moyens et lointains que nous n’étudions pas ici.

Théorie du capteur PIR

L’apparence physique du capteur PIR est très différente de celle des autres capteurs. Il comporte en effet une demi-boule en plastique translucide. C’est en fait une lentille de Fresnel qui concentre correctement les rayons infrarouges pour qu’ils viennent se rejoindre sur la toute petite surface du capteur qui est caché sous le dôme.

Voici l’aspect d’un capteur PIR avec son dôme reconnaissable (Figure 9.1).

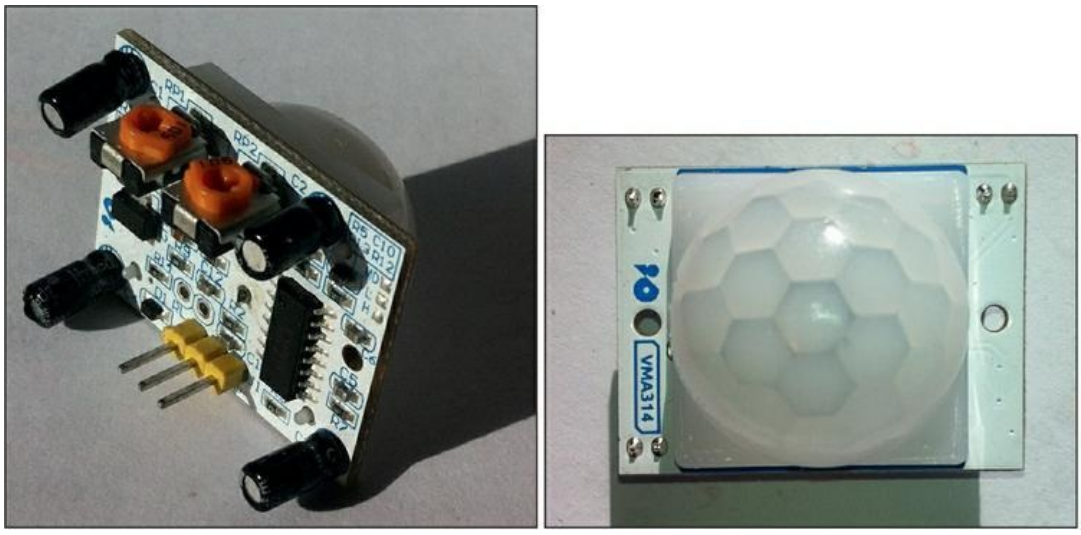


Figure 9.1 : Vue côté circuit et côté lentille du capteur PIR.

Le capteur fonctionne par comparaison des deux moitiés de son champ de détection, en découpant ce dernier en zones grâce à la lentille (Figure 9.2).

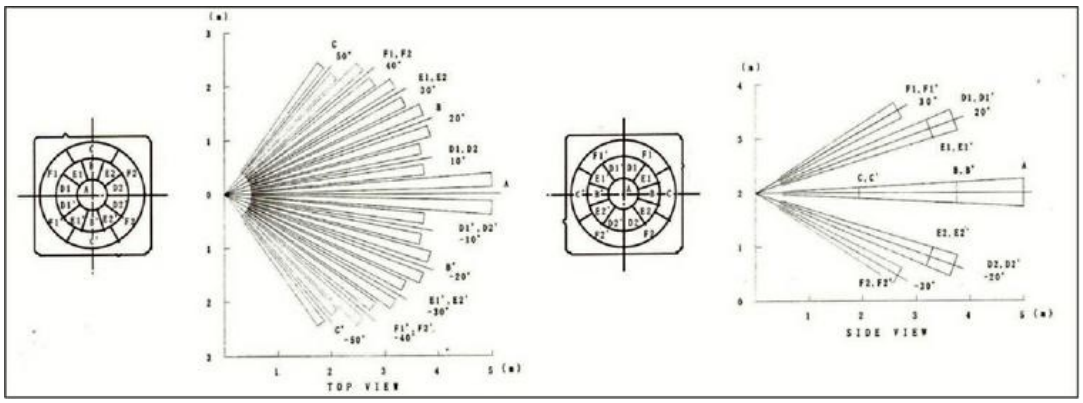


Figure 9.2 : La lentille de Fresnel concentre les rayons infrarouges.

Pratique du capteur PIR

Lorsqu'il est mis sous tension, il faut laisser au capteur quelques secondes pour qu'il dresse le tableau de son champ de détection. Rien ne doit bouger pendant cette phase de calibration. Une fois qu'elle est terminée, le capteur est capable de détecter la moindre différence par rapport à la situation qu'il a établie. C'est pour cette raison qu'il faut également lui laisser un petit temps de pause après chaque détection pour qu'il reconstruise son modèle.

Imagine que tu démarres le capteur puis que tu viennes t'asseoir dans son champ de vision. Il va détecter les mouvements qui correspondent à ton arrivée.

Si tu restes ensuite assis sans bouger pendant quelques secondes, le capteur va reconstruire l'image du champ de détection fixe. Cette nouvelle image va tenir compte de ton rayonnement infrarouge, et tu ne vas plus déclencher le capteur. Mais il te suffira de lever un doigt pour le déclencher ensuite.

Montage du circuit Gardien

Le montage du circuit du capteur est extrêmement simple puisqu'il suffit de relier ses trois broches à ta carte Arduino. Tu n'as même pas besoin de passer par une plaque d'essai. Ceci dit, je te conseille de l'utiliser, car tu auras besoin de cette plaque pour brancher un second composant dans la suite du chapitre.

Aucune résistance n'est à ajouter. Le capteur possède son propre régulateur de tension.

1. Prends un strap rouge mâle-femelle afin de pouvoir brancher le côté femelle dans la **broche marquée 5 V ou Vcc** du capteur. Branche le côté mâle n'importe où dans un des deux rails de distribution rouges marqués **+** ou **5 V** sur la plaque d'essai.

   Si ton capteur est doté de broches **femelles** pour ses sorties, utilise des straps mâle-mâle.

2. Branche la **broche du signal** (normalement la broche marquée OUT du milieu sur le capteur) avec un strap jaune, vert ou orange mâle-femelle entre le capteur et la **broche numérique 7** du connecteur DIGITAL sur la carte Arduino.

3. Branche un strap (noir de préférence ou marron) entre la **broche de masse GND** du capteur et un des deux rails de distribution bleus ou noirs marqués **–** ou **GND** sur la plaque d'essai.

4. Avec un strap mâle-mâle rouge, relie le rail +5 V de la plaque à la broche 5 V de la carte puis, avec un noir, le rail – (GND) à une des trois broches GND de la carte Arduino.

Le montage est terminé !

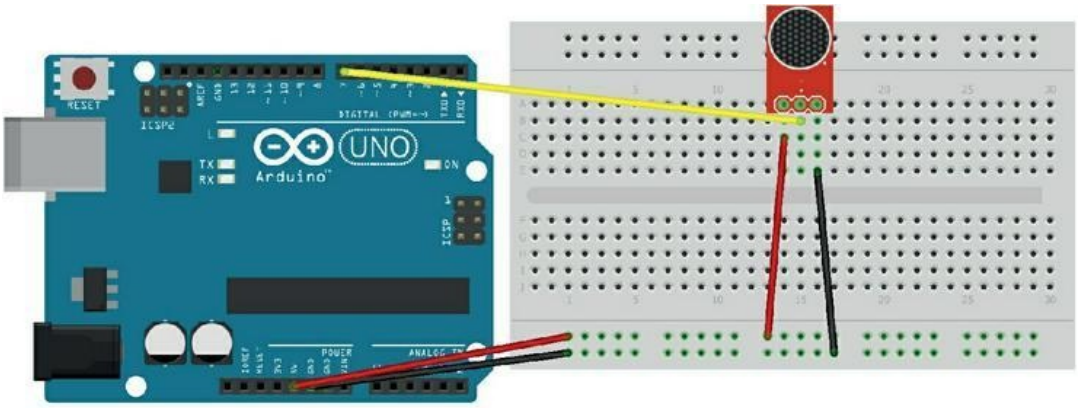


Figure 9.3 : Montage du projet Gardien.

L'écriture du texte source pourrait nous réserver quelques surprises. Nous n'allons donc pas nous jeter la tête la première dans la rédaction, classique erreur du débutant en programmation. Nous passons par une première phase d'étude de l'algorithme, c'est-à-dire des grandes lignes de la logique de fonctionnement.

L'algorithme de détection

1. Au départ, le capteur doit faire l'inventaire de ce qu'il voit et calculer l'équilibre entre les deux moitiés de son champ de surveillance.

   Le capteur demande quelques secondes (de 10 à 30) pour dresser ce portrait IR de la scène. C'est la calibration initiale (*étalonnage*).

2. Mais après chaque détection, la scène ayant sans doute changé, il faut à nouveau laisser le capteur refaire l'analyse de ce qu'il voit.

Il faut donc interdire la réaction à une nouvelle détection pendant ce temps. En effet, la détection par le capteur a bien lieu mais elle n'est pas exploitable. Il faut que le calme soit revenu pour détecter un vrai nouveau changement.

Nous allons avoir besoin d'une variable booléenne que l'on va armer (1) pour autoriser la détection, et désarmer (0) pour la suspendre. Nommons-la `bOKDetect`.

3. Il faut aussi démarrer un chronomètre dès qu'une détection a eu lieu, mais surtout ne pas le relancer tant qu'il n'y a plus de mouvement. Sinon, on n'arrive jamais au bout du temps de recalibration.

   Il faut donc prévoir une seconde variable booléenne que l'on va armer (mettre à 1) pour forcer le redémarrage du chronomètre, et désarmer pour l'interdire. Je la nomme `bRedemarrerKrono`.

4. Pour mesurer l'écoulement du temps de recalibration, nous prévoyons une variable spacieuse (du type long) qui va mémoriser un nombre de secondes depuis le démarrage du programme en interrogeant la fonction prédéfinie `millis()`. Pour savoir si le temps de recalibration est écoulé, il suffira d'interroger à nouveau cette fonction.

   Par exemple, si le chrono a été démarré à 15 secondes, avec une pause de 5 secondes, la détection sera de

nouveau autorisée seulement quand l'appel à `millis()` renverra au moins la valeur 21.

5. La donnée qui va déterminer la logique d'action est l'état de la broche de signal du capteur PIR.

   Si elle est à 1 (HIGH), il y a eu Mouvement.

   Si elle est à 0 (LOW), c'est la situation de Calme plat.

6. Voici les blocs conditionnels à écrire :

Situation de calme plat

a. **SI** Calme plat **ET SI** Détection suspendue **ET SI** Temps de recalibration écoulé :

   Armer la détection.

b. **SI** Calme plat **ET SI** Détection suspendue **MAIS** Temps de recalibration pas écoulé :

   Ne rien faire (Laisser la détection suspendue).

c. **SI** Calme plat **ET SI** Redémarrage chrono autorisé (un mouvement vient d'être détecté) :

   Lancer le chronomètre de la pause de recalibration.

   Empêcher son redémarrage avant la prochaine détection.

Situation de mouvement détecté

a. **SI** Mouvement détecté **MAIS** Détection suspendue (pendant la recalibration)

Autoriser Redémarrage chrono de pause (prolonge la recalibration).

b. **SI** Mouvement détecté **ET SI** Détection armée :

Déclencher l'alarme.

Suspendre la détection.

Autoriser Redémarrage chrono de recalibration.

Lancer le chrono consiste à récupérer l'heure actuelle comme heure de départ de la recalibration grâce à la fonction standard `millis()`.

La Figure 9.4 montre l'algorithme correspondant.

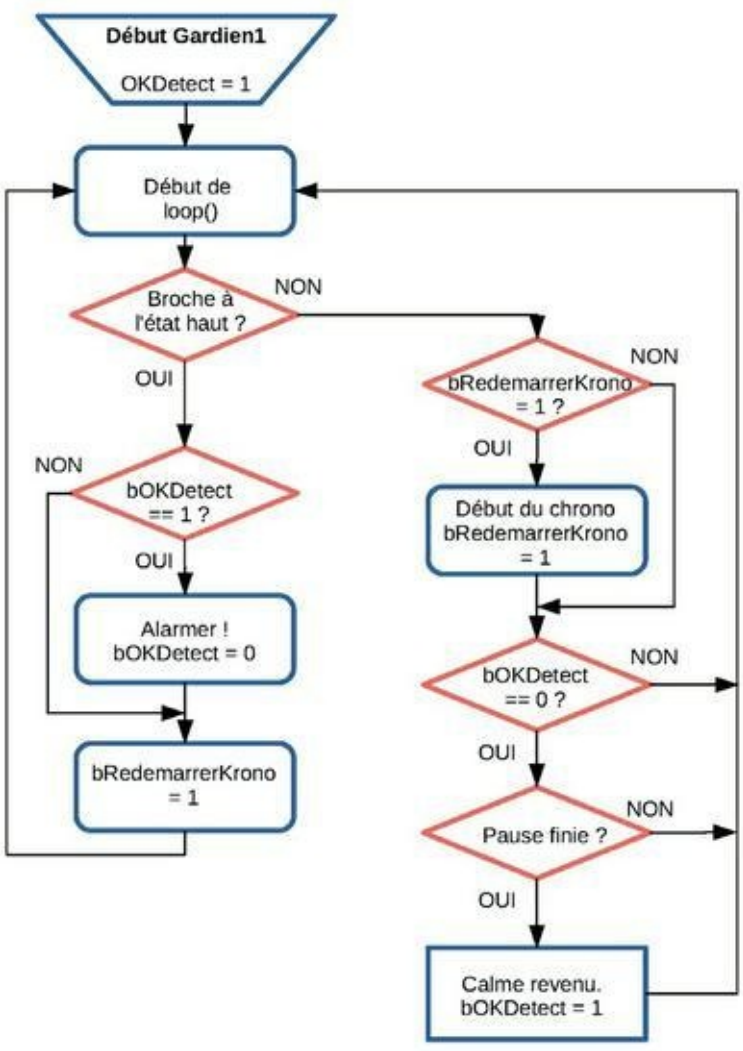


Figure 9.4 : Algorithme d'exploitation du capteur PIR.

Rédigeons le texte source

Grâce à notre étude préalable, l'écriture va être assez facile. Nous allons définir quelques variables globales, écrire le code de calibration dans la fonction `setup()` puis, dans la boucle principale, prévoir deux gros blocs conditionnels :

» le premier bloc pour le cas où rien n'a été détecté ;

» le second bloc lorsque le capteur vient de détecter un mouvement.

Voici d'abord le texte source complet tel que je te propose de le saisir (ou de le récupérer dans l'archive des exemples si cela te paraît trop long). Tu peux choisir comme nom de projet `Gardien`.

Je rappelle qu'il est conseillé de prendre le temps de saisir le code pour encore mieux le maîtriser.

Listing 9.1 : Version 1 du projet Gardien

```
// Gardien1
// OEN170321

const byte yBroPIR = 7;
const byte yBroLED = 13;
const byte yTempsKalib = 10;        // Max 30
secondes

unsigned long gDebutKrono;
unsigned long gPause = 1000;        //  Max
5000 millis
boolean bOKDetect = true;
boolean bRedemarrerKrono;

////////////////////////////
void setup(){
  Serial.begin(9600);
  pinMode(yBroPIR, INPUT);
  pinMode(yBroLED, OUTPUT);
  digitalWrite(yBroPIR, LOW);
```

```
  Serial.println("** Projet GARDIEN.
Calibration du senseur.");
  Serial.print("** Que personne ne bouge. On
patiente. ");
    for(int i = 0; i < yTempsKalib; i++) {
      delay(1000);
      Serial.print(".");
    }
    Serial.println("C'est parti ! ");
    delay(50);
  }

//////////////////////////
void loop(){
  if(digitalRead(yBroPIR) == LOW) {
    if(bRedemarrerKrono) {
      gDebutKrono = millis();
      bRedemarrerKrono = false;
    }
    if(!bOKDetect && ( (millis() -
gDebutKrono) > gPause) ) {
      bOKDetect = true;
      delay(50);
      bavarder(1);
    }
  }

  if(digitalRead(yBroPIR) == HIGH) {
    if(bOKDetect) {
     bavarder(0);
      Serial.println("Alarme ! Alarme !
```

```
Alarme ! ");
      delay(50);
      bOKDetect = false;
    }
    bRedemarrerKrono = true;
  }
}

void bavarder(byte bType) {
  if (bType == 0) {
    Serial.println("---");
    Serial.print("Quelque chose bouge.
Instant : ");
    Serial.print(millis()/1000);
    Serial.println(" secondes");
  }
  if (bType == 1) {
      Serial.print("Le calme est revenu au
bout de : ");
      Serial.print((millis() -
gPause)/1000);
      Serial.println(" secondes");
  }
}
```

En route pour la visite guidée :

1. En début de programme, nous déclarons trois constantes pour les deux broches et pour la durée de calibration. Cette durée est à choisir entre 10 et 30 secondes, selon le modèle exact du

capteur dont tu disposes. Une valeur de 10 convient généralement.

2. Nous définissons ensuite deux variables très spacieuses, du type unsigned long, ce qui permet de compter jusqu'à 4 milliards de millisecondes, soit environ 45 jours sans mouvement avant que le chronomètre reparte de 0 :

```
unsigned long gDebutKrono ;
unsigned long gPause = 1000 ; // Max 5000
millis
```

La deuxième variable du type long aurait pu être plus compacte puisque nous n'y stockons pas de valeur supérieure à 5000. Nous aurions même pu la choisir de type const. Pour que ce soit cohérent avec sa collègue, nous avons uniformisé le type. Au diable, les varices !

Tu constates au passage que la durée de recalibration est plus faible que celle du premier démarrage.

3. Nous trouvons ensuite les deux variables de contrôle du type boolean dont nous avons parlé dans l'étude de l'algorithme. Nous donnons à la première la valeur initiale true (**1**) afin d'être prêts dès le départ à détecter un mouvement.

```
boolean bOKDetect = true ;
```

```
boolean bRedemarrerKrono ;
```

4. Nous arrivons ensuite dans la fonction de préparation `setup()`. La boucle de répétition `for` permet d'imposer un délai de dix fois une seconde pour laisser le temps au capteur d'inventorier son champ de vision :

```
for(int i = 0 ; i < yTempsKalib ; i++) {
delay(1000) ;
Serial.print(".") ;
}
```

5. Nous arrivons alors dans la boucle principale. On voit clairement qu'il y a deux blocs conditionnels. Chacun contient un ou deux sous-blocs conditionnels.

6. Le premier bloc conditionnel gère l'absence de mouvement :

```
if(digitalRead(yBroPIR) == LOW) {
```

Dans ce cas, nous commençons une nouvelle phase de recalibration seulement si la variable `bRedemarrerKrono` le permet :

```
 if(bRedemarrerKrono) {
   gDebutKrono = millis();
    bRedemarrerKrono = false;

  }
```

7. Le second test est plus compliqué car il est hybride :

```
if( ! bOKDetect && ( (millis() -
gDebutKrono) > gPause) ) {
```

Puisque la condition réunit deux expressions par l'opérateur `&&` (ET), les deux doivent être vraies (valeur différente de 0) pour que le test passe. Ce test ne réussit que si :

a. la détection n'est PAS autorisée (`!bOKDetect`) ;

   ET SI

b. simultanément, le temps de recalibration minimal s'est écoulé. Pour le savoir :

   nous soustrayons au nombre de secondes actuel (le temps écoulé tel que renvoyé par la fonction standard `millis()`) le top départ stocké dans la variable du chronomètre ;

   nous comparons cette valeur au temps de pause de recalibration minimal.

```
( (millis() - gDebutKrono) > gPause)
```

On doit avoir autant de parenthèses ouvrantes que de parenthèses fermantes.

Si le test réussit, nous autorisons à nouveau la détection :

```
bOKDetect = true;
```

Tu remarques l'appel à une fonction `bavarder()` avec le paramètre égal à 1. Cette ligne peut être mise en

commentaires si tu n'as pas besoin de l'affichage. Pour le moment, c'est le seul moyen que nous ayons de savoir que l'alarme se déclenche.

8. L'autre bloc conditionnel gère le cas d'une détection par le capteur. Si un mouvement a été détecté, nous ne déclenchons pas obligatoirement l'alarme, mais seulement si la détection a été autorisée. Si elle ne l'est pas, nous nous contentons de faire redémarrer le chronomètre, car un nouveau mouvement s'est présenté alors que nous étions déjà en phase de reconstruction de l'image. Jacques a dit : « Pas bouger ! »

   Ici aussi, nous appelons la fonction `bavarder()` avec le paramètre égal à 0.

9. Nous arrivons enfin à cette fonction locale qui s'occupe de tous les affichages, en cas de recalibration et de détection.

Test du projet

Une fois le programme saisi ou chargé, vérifie toutes les instructions puis lance un téléversement après avoir branché la carte Arduino.

Dès que la compilation a réussi, ouvre la fenêtre du Moniteur série. À partir de ce moment, ne bouge plus du tout. Observe le passage de la phase de calibration (Figure 9.5).

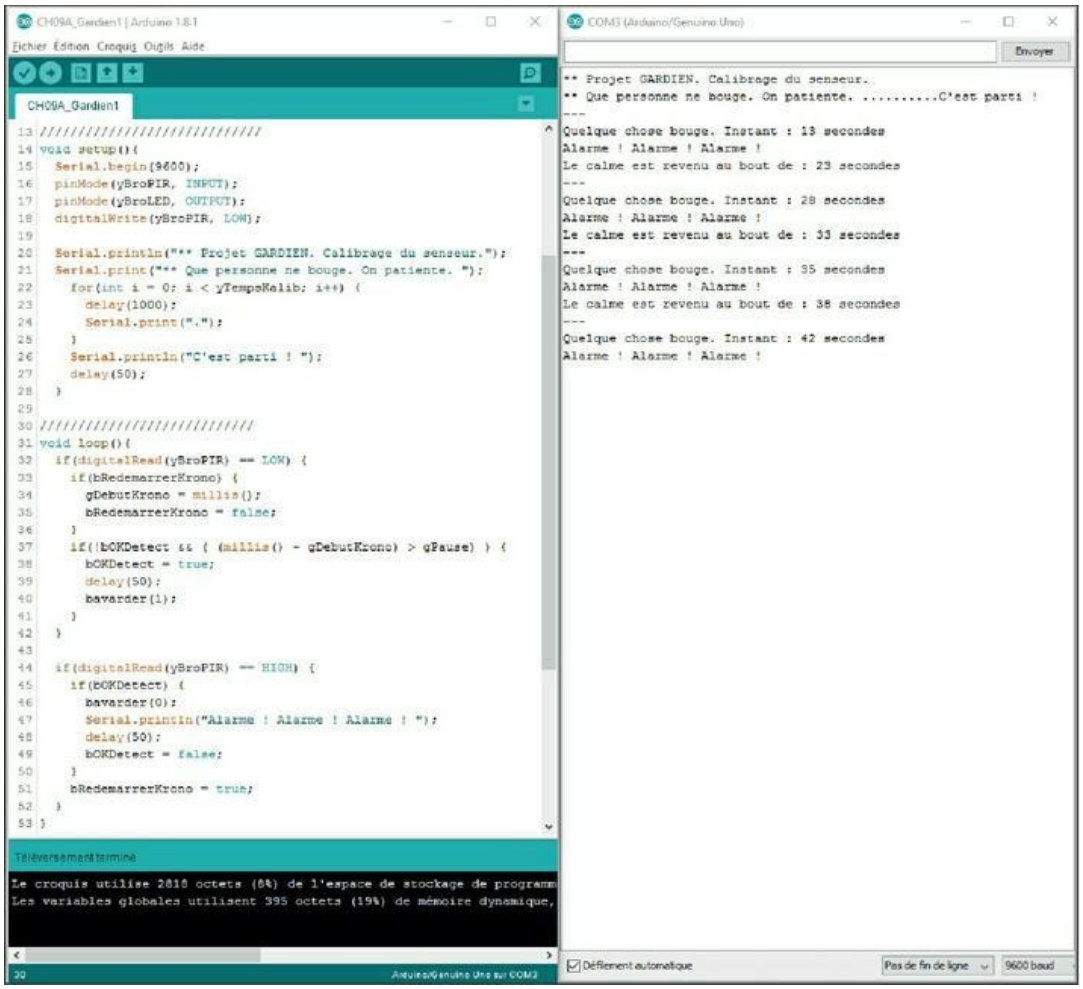
Ne t'inquiète pas si la détection ne semble pas fonctionner immédiatement après la fin de la calibration. Il faut parfois un peu plus de temps au capteur, d'autant que cela dépend également du

réglage des deux potentiomètres qui sont normalement présents sur le circuit du capteur.

Réglages fins

L'un des potentiomètres du capteur sert à régler la portée de la détection qui peut aller de quelques centimètres à 6 m environ. L'autre potentiomètre sert à régler manuellement le délai minimal avant la prochaine détection. A priori, il n'est pas nécessaire de toucher à ces réglages, sauf s'ils ont été modifiés par erreur. Dans ce cas, tu feras des essais par tâtonnements successifs.

Notre capteur fonctionne donc correctement, mais dans cette première version, nous ne faisons qu'afficher un message dans le Moniteur série. Voyons donc comment utiliser un haut-parleur pour faire retentir une véritable alarme sonore.


CH09A_Gardien1 | Arduino 1.8.1
Fichier Édition Croquis Outils Aide
CH09A_Gardien1
COM3 (Arduino/Genuino Uno)
Envoyer
** Projet GARDIEN. Calibrage du senseur.
** Que personne ne bouge. On patiente. .....C'est parti !

Quelque chose bouge. Instant : 13 secondes
Alarme ! Alarme ! Alarme !
Le calme est revenu au bout de : 23 secondes

Quelque chose bouge. Instant : 28 secondes
Alarme ! Alarme ! Alarme !
Le calme est revenu au bout de : 33 secondes

Quelque chose bouge. Instant : 35 secondes
Alarme ! Alarme ! Alarme !
Le calme est revenu au bout de : 38 secondes

Quelque chose bouge. Instant : 42 secondes
Alarme ! Alarme ! Alarme !
Téléversement terminé
Le croquis utilise 2818 octets (8%) de l'espace de stockage de programm
Les variables globales utilisent 395 octets (19%) de mémoire dynamique,
Défilement automatique
Pas de fin de ligne
9600 baud

Figure 9.5 : Exécution de Gardien1.

Une alarme audio

Le son est un phénomène analogique : une corde ou une peau de tambour vibrent comme une balançoire jusqu'à revenir à l'état de repos. Comment générer du son avec une machine numérique ?

Faites du bruit !

Sur un tambour : la peau est tendue. Quand on la frappe, elle est repoussée en arrière. Du fait qu'elle est élastique, elle revient un peu trop, puis repart en arrière, etc.

Dans un haut-parleur, l'équivalent de la peau est la membrane. Elle est maintenue en place par un aimant en forme de bâton. Cet aimant est permanent. Il crée un champ magnétique sans avoir besoin d'être alimenté.

On enroule sans serrer autour du bâton une bobine de fil et on fixe la membrane en papier à cette bobine. Les deux bouts de fil sont reliés à deux bornes isolées.

Quand du courant passe dans la bobine, cela crée un champ magnétique qui fait coulisser la bobine et donc la membrane en papier sur le bâton, ce qui repousse l'air de part et d'autre. Le résultat s'entend, comme les vibrations d'une peau de tambour.

Chaque aller-retour de la bobine génère une onde dans l'air, comme une vaguelette à la surface de l'eau. Si la bobine fait 440 allers-retours en une seconde, on dit que la fréquence du son est de 440 Hz (hertz).

L'unité « hertz » a été choisie en l'honneur du chercheur allemand Heinrich Rudolf Hertz (1857-1894).

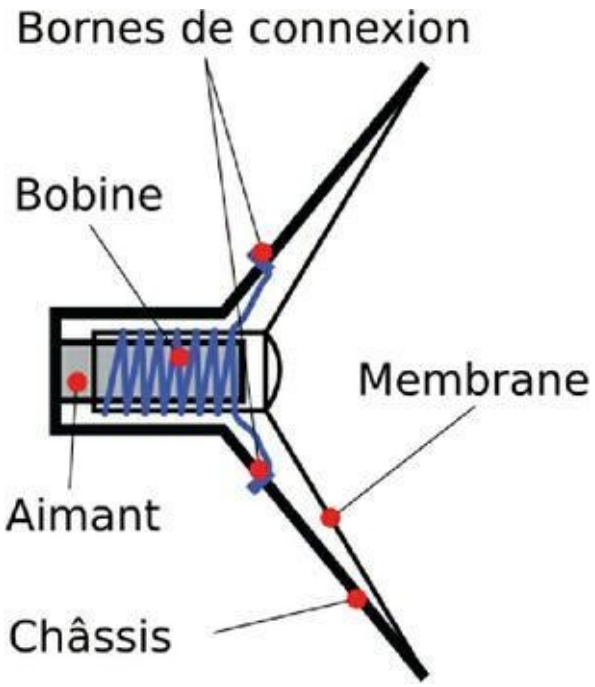


Figure 9.6 : Principe d'un haut-parleur.

En faisant alterner l'état d'une broche de sortie numérique de notre mikon, on doit pouvoir générer un son ? C'est ce que nous allons prouver.

La Figure 9.7 montre l'onde telle qu'enregistrée avec la première version de notre projet. L'outil utilisé est **Audacity**.

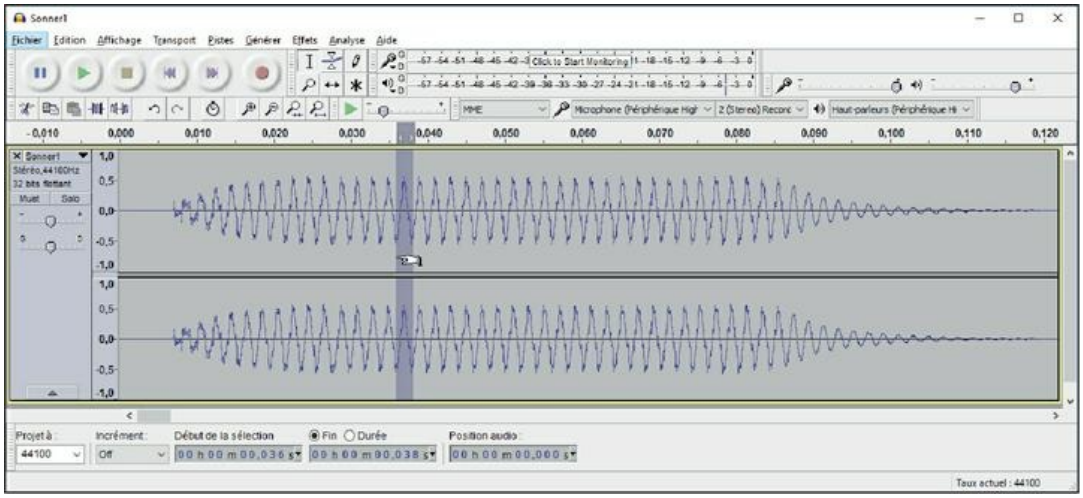


Figure 9.7 : Visualisation d'une onde acoustique.

Commençons par construire le circuit.

Montage de Sonner

Pour faire nos tests, nous n'avons besoin que d'un petit haut-parleur et d'une résistance de 100 à 270 ohms.

Une des valeurs-clés des haut-parleurs est l'impédance. C'est un peu comme une résistance et on utilise d'ailleurs la même unité, l'ohm. La différence est que cette impédance varie en fonction de la fréquence. La plupart des petits haut-parleurs ont une toute petite impédance de 8 ohms. Il faut donc ajouter une résistance pour protéger le mikon. Une valeur de 220 ou 270 ohms conviendra. Cela va énormément limiter le courant dans la bobine du haut-parleur, mais il restera assez d'énergie pour que tu l'entendes.

Si tu récupères un des haut-parleurs des oreillettes d'un casque stéréo, l'impédance est souvent de 32 ohms. Tu peux dans ce cas

choisir une résistance plus faible, par exemple de 100 ohms.

Voici comment monter ce circuit :

1. Si le haut-parleur n'est pas muni de deux fils soudés, et puisque nous ne faisons aucune soudure, procède ainsi :

   a. Sacrifie deux straps avec au moins un côté mâle, un rouge et un noir. Pour chacun d'eux, coupe le connecteur d'un côté, dénude le fil sur 1 cm puis entortille-le sur lui-même.

   b. Fais passer le bout d'un fil dans une oreille de connexion du haut-parleur (Figure 9.8), fais une boucle et entortille les deux brins pour que la connexion tienne. Bien sûr, si tu tires dessus, cela ne résistera pas.

   c. Fais de même pour l'autre fil.

Figure 9.8 : Les bornes de connexion du haut-parleur.

2. Installe la résistance de 220 ou 270 ohms sur la plaque d'essai, une patte dans le rail du négatif (0 V) et l'autre patte dans une rangée libre.
3. Connecte le rail du négatif de la plaque à l'un des connecteurs de masse GND de la carte Arduino.
4. Branche le fil noir du haut-parleur dans la rangée de la plaque où arrive la résistance.
5. Branche le fil rouge du haut-parleur à la *sortie numérique 8* du connecteur DIGITAL de la carte Arduino.

Le montage est terminé (Figure 9.9).

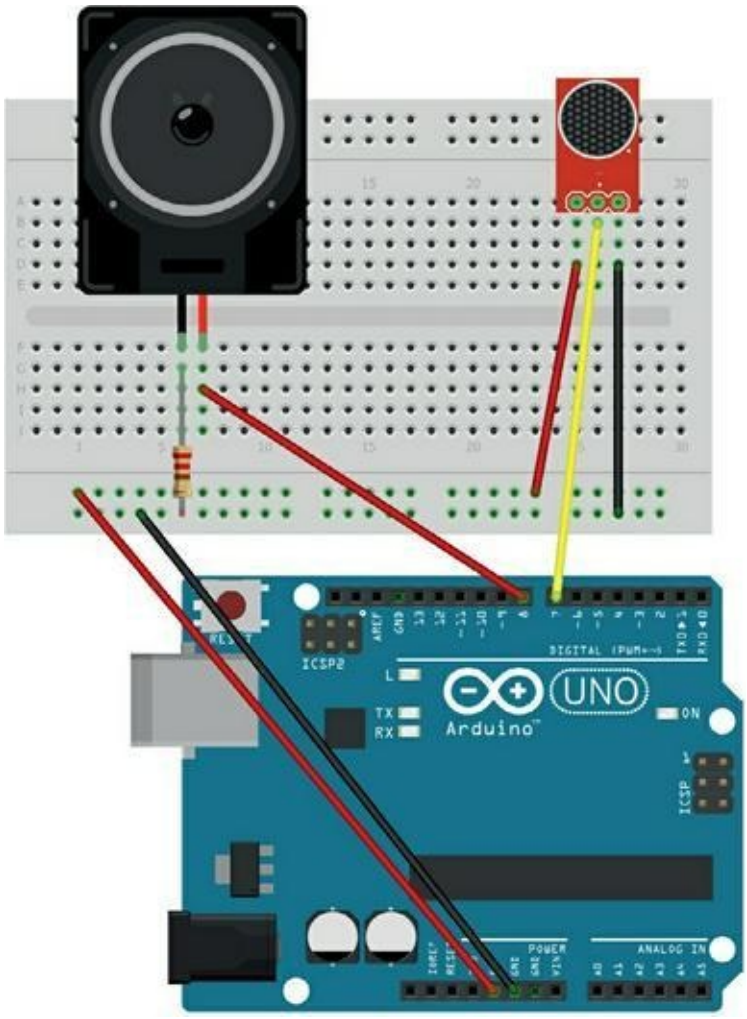


Figure 9.9 : Montage du projet Sonner.

Pousser l'air avec des 0 et des 1 !

Pour déplacer la bobine, il nous suffit d'envoyer avec `digitalWrite()` une alternance de 1 et de 0 sur la broche de sortie. Ce qui va décider de la fréquence de la note (sa hauteur) est le temps pendant lequel nous allons rester dans le même état haut ou bas.

Nous allons écrire une boucle de répétition dans ce style :

1. Mettre la sortie à l'état 1 (du courant passe dans la bobine qui se déplace).

2. Attendre un tout petit peu avant de basculer.
3. Mettre la sortie à l'état 0 (la bobine est ramenée au repos par l'aimant).
4. Attendre un tout petit peu avant de rebasculer.

Les étapes d'attente 2 et 4 vont déterminer la hauteur. Au lieu d'utiliser la fonction `delay()`, bien connue maintenant, nous adoptons sa variante mille fois plus précise, `delayMicroseconds()`.

```
    digitalWrite(iBroSON, 1) ;
     delayMicroseconds(hauteur) ;
     digitalWrite(iBroSON, 0) ;
delayMicroseconds(hauteur) ;
```

Ces quatre étapes vont être répétées au moins cent fois pour que le son soit bien audible (à moins de 10 tours, on n'entend plus qu'un plop).

Codage de Sonner1

Nous déclarons deux variables de type `int` dans la boucle principale :

```
iHauteur = 800 ;
iDuree   = 100 ;
```

La première contient des millisecondes et l'autre, un nombre de tours de boucle de répétition.

Après l'émission d'un bip, nous imposons un délai pour le séparer du suivant.

Listing 9.2 : CH09B_Sonner1.ino

```
// Sonner1
// OEN170325

const int yBroSON = 8;

void setup() {
  pinMode(yBroSON, OUTPUT);
}

void loop() {
  int iHauteur = 800;
  int iDuree   = 100;
  for (int i = 0; i < iDuree; i++) {
    digitalWrite(yBroSON, 1);
    delayMicroseconds(iHauteur);
    digitalWrite(yBroSON, 0);
    delayMicroseconds(iHauteur);
  }
  delay(200); // Entre 2 bips
}
```

Après avoir saisi ce programme, vérifie si nécessaire les branchements, puis téléverse.

Si le son est trop fort, augmente la valeur de la résistance (sans oublier de débrancher d'abord !).

Nous venons de créer un son typique des alarmes de tous les appareils numériques. Rapidement énervant, non ?

Au lieu de gérer nous-mêmes la production du son, voyons comment profiter d'une fonction standard installée en même temps que l'atelier Arduino : la fonction `tone()`.

Sonne, oh tone()

La fonction `tone()` et sa collègue `noTone()` sont définies dans une librairie standard. Il n'y a donc pas besoin d'ajouter une mention `#include "NomLibrairie"` au début de notre projet.

Les deux différences majeures avec `digitalWrite()` sont celles-ci :

- nous pouvons directement indiquer en hertz la hauteur de la note à jouer ;
- la note continue à être émise en continu. C'est pourquoi il faut utiliser l'autre fonction, `noTone()`, pour retrouver le silence.

Codage de Sonner2

Listing 9.3 : CH09B_Sonner2.ino

```
// Sonner2
// OEN170325

const byte yBroSON = 8;

void setup() {
  pinMode(yBroSON, OUTPUT);
}

void loop() {
  int iHauteur = 440;         // Le LA des
musiciens
```

```
  int iDuree   = 200;
  sonner(iHauteur, iDuree);
  delay(500);                    // Silence
entre notes
}

void sonner(int iFreq, int iLongueur) {
  tone(yBroSON, iFreq);
  delay(iLongueur);
  noTone(yBroSON);
}
```

Lance un test. Le résultat sonore est quasiment identique. Dorénavant, bien des choses deviennent envisageables. Par exemple :

- Pour `iHauteur` et `iDuree`, tu peux définir deux tableaux de valeurs contenant les hauteurs et les durées des notes d'une mélodie.
- Tu peux profiter d'un fichier de définition nommé `pitches.h` qui contient une équivalence entre les fréquences et des noms de la plupart des notes audibles. Il faut dans ce cas ajouter une directive `#include "pitches.h"` en début de fichier et placer une copie de ce fichier dans le même sous-dossier que le texte source.

Allons un peu plus loin en concevant une animation sonore pour un signal d'alarme.

Une sirène sans écailles

La musique et les mathématiques sont très liées. Pendant des millénaires, les quatre matières principales à l'école (le quadrivium) étaient :

- l'arithmétique,
- la géométrie,
- la musique,
- l'astronomie.

Le son et les chiffres font très bon ménage. Et le mikon adore les chiffres !

Pour concevoir une vraie sirène qui monte dans les aigus et redescend, il suffit de créer deux boucles de répétition :

- une boucle qui monte de hertz en hertz ;
- une seconde boucle qui descend de hertz en hertz.

Cela va constituer un audiomotif !

Passons directement à la mise en pratique. Je propose de balayer la plage de fréquences entre le la du diapason et 2000 Hz.

Listing 9.4 : CH09B_Sonner3.ino

```
// Sonner3
// OEN170325

const byte yBroSON   = 8;

void setup() {
  pinMode(yBroSON, OUTPUT);
}
void loop() {
```

```
  alarmer();
//  delay(500);    // Facultatif
}

void alarmer() {
  const int  iBasse  = 440;
  const int  iHaute = 2000;
  for (int i = iBasse; i < iHaute; i++) {
    tone(yBroSON, i);
    delay(1);
  }
  for (int i = iHaute; i > iBasse; i--) {
    tone(yBroSON, i);
    delay(1);
  }
  noTone(yBroSON);
}
```

Test de Sonner3

J'imagine que tu devines quel résultat tes oreilles vont pouvoir vérifier : une vraie sirène d'alarme ! Dans la prochaine section, nous allons réinjecter la fonction

Commentaires sur Sonner3

Dans la fonction `alarmer()`, nous trouvons deux blocs de répétition comme prévu. Tu constates que j'ai déclaré les deux variables `iBasse` et `iHaute` (des constantes en fait) de façon locale et non en début de programme comme d'habitude. Cela a plusieurs conséquences :

- Ces variables n'existent que dans la fonction ; si tu tentes de les citer ailleurs, par exemple dans `loop()`, le compilateur va te répondre qu'il ne les connaît pas. Les variables appartiennent à la fonction, un peu comme les données d'un objet vues dans le Chapitre 8.
- Si tu définis une variable globale homonyme, elle sera masquée par la variable locale.
- En copiant-collant le bloc de la fonction `alarmer()`, nous emportons avec nous tout ce qu'il lui faut pour travailler.

Retour au gardien

Puisque nous disposons d'une fonction d'alarme audio, voyons comment la réinjecter dans le projet Gardien du début de chapitre.

Nous allons injecter la sirène depuis le projet Sonner3, c'est-à-dire le corps de la fonction `alarmer()`.

Listing 9.5 : Version 2 du projet Gardien

```
// Gardien2
// OEN170321

const byte yBroPIR = 7;
const byte yBroLED = 13;
const byte yTempsKalib = 10;      // Max 30
secondes
const byte yBroSON = 8;            // AJOUT
```

```
unsigned long gDebutChrono;
unsigned long gPause = 100;    //  Max 5000
boolean bOKDetect = true;
boolean bRedemarrerKrono;

////////////////////////////
void setup(){
  Serial.begin(9600);
  pinMode(yBroPIR, INPUT);
  pinMode(yBroLED, OUTPUT);
  pinMode(yBroSON, OUTPUT);             //
AJOUT
  digitalWrite(yBroPIR, LOW);

  Serial.println("** Projet GARDIEN.
Calibrage du senseur.");
  Serial.print("** Que personne ne bouge. On
patiente. ");
    for(int i = 0; i < yTempsKalib; i++) {
      delay(1000);
      Serial.print(".");
      tone(yBroSON, 440, 200);         //
AJOUT
    }
    Serial.println("C'est parti ! ");
    delay(50);
  }

////////////////////////////
void loop(){
  if(digitalRead(yBroPIR) == LOW) {
    if(bRedemarrerKrono) {
```

```
      gDebutChrono = millis();
      bRedemarrerKrono = false;
    }
    if(!bOKDetect && ( (millis() -
gDebutChrono) > gPause) ) {
      bOKDetect = true;
      delay(50);
      bavarder(1);
    }
  }

  if(digitalRead(yBroPIR) == HIGH) {
    if(bOKDetect) {
      bavarder(0);
      Serial.println("Alarme ! Alarme !
Alarme ! ");
      alarmer();
// AJOUT
      delay(50);
      bOKDetect = false;
    }
    bRedemarrerKrono = true;
  }
}

void bavarder(byte bType) {
  if (bType == 0) {
    Serial.println("---");
    Serial.print("Quelque chose bouge.
Instant : ");
    Serial.print(millis()/1000);
    Serial.println(" secondes");
```

```
  }
  if (bType == 1) {
      Serial.print("Le calme est revenu.
Instant : ");
      Serial.print((millis() -
gPause)/1000);
      Serial.println(" secondes");
  }
}

//// NOUVEAU
void alarmer() {
  const int  iBasse  = 440;
  const int  iHaute = 2000;
  for (int i = iBasse; i < iHaute; i++) {
    tone(yBroSON, i);
    delay(1);
  }
  for (int i = iHaute; i > iBasse; i--) {
    tone(yBroSON, i);
    delay(1);
  }
  noTone(yBroSON);
}
```

Retouches de Gardien2

Nous insérons quatre nouvelles instructions dans le code existant et collons tout à la fin du bloc de la fonction `alarmer()` depuis Sonner3.ino.

Les insertions sont marquées par le commentaire de ligne // AJOUT. Voici ces ajouts :

- Tout au début, une déclaration pour la broche du haut-parleur.
- Dans `setup()`, une instruction pour préparer la broche du haut-parleur en sortie.
- Toujours dans `setup()`, une instruction pour émettre un bip :

  ```
  tone(yBroSON, 440, 200) ;
  ```

  Si tu observes bien, il y a un troisième paramètre (200). Il est facultatif et permet de limiter la durée d'émission du son.
- Enfin, en cas de détection autorisée, nous appelons notre nouvelle fonction pour siréner de bon cœur !

Ceci termine le projet d'alarme. Avant de clore le chapitre, je te propose un test d'audition gratuit.

Testons nos oreilles

Le petit projet bonus qui suit est à manier avec précaution :

1. Il émet des sons très aigus. Mieux vaut éloigner les chiens que cela pourrait énerver.
2. Nous utilisons un petit haut-parleur à tout faire qui est incapable d'émettre des sons très aigus aussi bien que les sons du milieu du spectre sonore, soit de 200 Hz à 4000 Hz.

3. Ne laisse pas le programme fonctionner en continu.

Le principe consiste à partir d'une note de base et à monter en fréquence en ajoutant à la note courante la valeur de la note de base. Tu vois qu'elle est définie en tant que synonyme par #define.

Listing 9.6 : CH09C_TestOreilles.ino

```
// TestOreilles
// OEN170325

#define DIAPA 440
int yBroSON = 8;

void setup() {
  pinMode(yBroSON, OUTPUT);
  Serial.begin(9600);
  Serial.println("Attention les oreilles ! !
! ");
}

void loop() {
  int iNote;
  for (iNote = DIAPA; iNote < 10000; iNote
+= DIAPA) {
    tone(yBroSON, iNote);
    Serial.print("On entend encore ceci ?  :
");
    delay(800);
    Serial.print(iNote);
    Serial.println();
  }
```

```
  noTone(yBroSON);
}
```

Testons le test

Pour faire le test, il suffit d'écouter jusqu'à quelle fréquence tu entends la note (Figure 9.10).

Attention : ce test n'est pas du tout scientifique. Il procure néanmoins une idée de ce vaste monde des fréquences aiguës qui donnent leur couleur aux sons musicaux, tous plus graves. Les aigus sont les sons harmoniques.

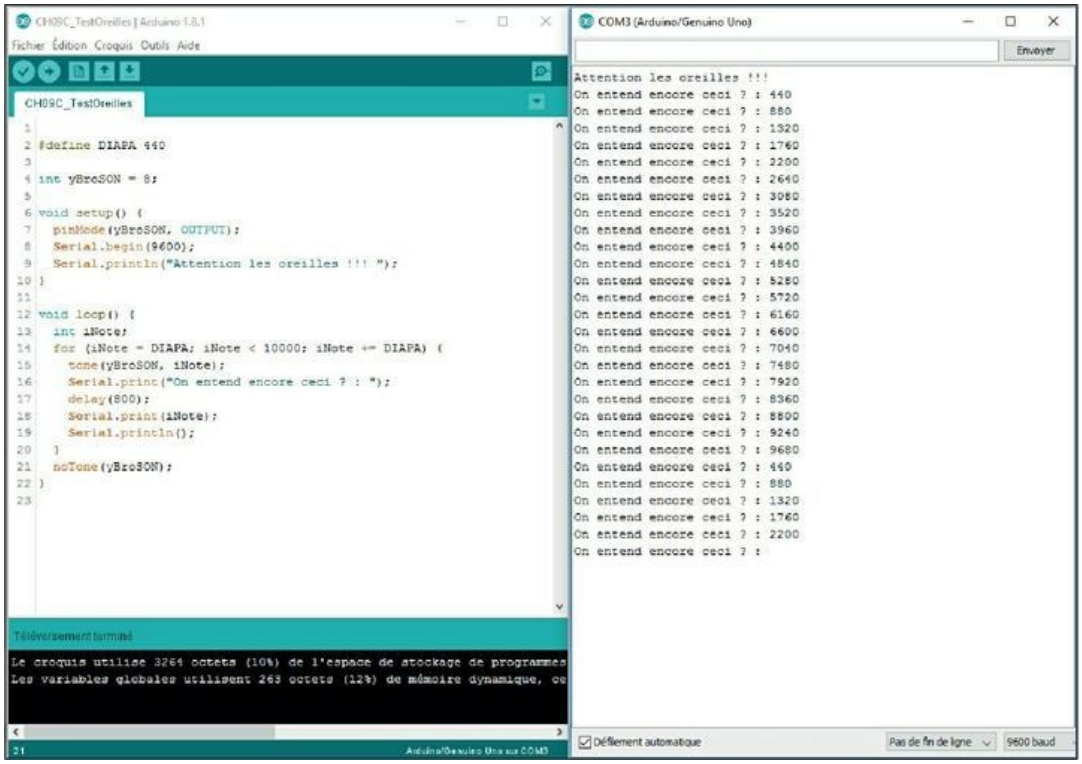

CH09C_TestOreilles | Arduino 1.8.1
Fichier Édition Croquis Outils Aide
CH09C_TestOreilles
#define DIAPA 440
int yBroSON = 8;
void setup() {
pinMode(yBroSON, OUTPUT);
Serial.begin(9600);
Serial.println("Attention les oreilles !!! ");
}
void loop() {
int iNote;
for (iNote = DIAPA; iNote < 10000; iNote += DIAPA) {
tone(yBroSON, iNote);
Serial.print("On entend encore ceci ? : ");
delay(800);
Serial.print(iNote);
Serial.println();
}
noTone(yBroSON);
}
Téléversement terminé
Le croquis utilise 3264 octets (10%) de l'espace de stockage de programmes
Les variables globales utilisent 263 octets (12%) de mémoire dynamique, co
COM3 (Arduino/Genuino Uno)
Envoyer
Attention les oreilles !!!
On entend encore ceci ? : 440
On entend encore ceci ? : 880
On entend encore ceci ? : 1320
On entend encore ceci ? : 1760
On entend encore ceci ? : 2200
On entend encore ceci ? : 2640
On entend encore ceci ? : 3080
On entend encore ceci ? : 3520
On entend encore ceci ? : 3960
On entend encore ceci ? : 4400
On entend encore ceci ? : 4840
On entend encore ceci ? : 5280
On entend encore ceci ? : 5720
On entend encore ceci ? : 6160
On entend encore ceci ? : 6600
On entend encore ceci ? : 7040
On entend encore ceci ? : 7480
On entend encore ceci ? : 7920
On entend encore ceci ? : 8360
On entend encore ceci ? : 8800
On entend encore ceci ? : 9240
On entend encore ceci ? : 9680
On entend encore ceci ? : 440
On entend encore ceci ? : 880
On entend encore ceci ? : 1320
On entend encore ceci ? : 1760
On entend encore ceci ? : 2200
On entend encore ceci ? :
Défilement automatique
Pas de fin de ligne
9600 baud

Figure 9.10 : Le test d'audition.

La note la plus aiguë d'un piano a une fréquence d'environ 4400 Hz.

Récapitulons

Dans ce chapitre, nous avons vu :

- comment utiliser un capteur de mouvement PIR ;
- ce qu'est un algorithme ;
- comment utiliser un haut-parleur.

Chapitre 10
Le voleur de couleurs

AU MENU DE CE CHAPITRE :

- **Trois diodes LED en une**
- **Mélangeons les teintes**
- **Le capteur de couleurs**

Nous venons d'apprendre à produire de façon contrôlée des ondes sonores qui sont des ondes mécaniques. Voyons maintenant comment faire la même chose avec des ondes électriques : créer n'importe quelle nuance de lumière visible et reconnaître une nuance pour la reproduire (plus ou moins fidèlement).

Des ondes lumineuses

Presque tout ce que tu vois avec tes yeux est d'une certaine couleur parce que la matière mange la lumière. Quand un objet est éclairé avec de la lumière blanche (transparente plutôt), il avale certaines couleurs et n'en renvoie qu'une bien précise qui devient la couleur de cet objet. Un objet est rouge parce qu'il soustrait toutes les autres couleurs sauf le rouge. C'est la synthèse soustractive.

Figure 10.1 : Effet de la synthèse soustractive.

Plus les tons de peinture que tu mélanges sont nombreux, plus le résultat s'approche d'un brun presque noir. Les matières se combinent pour ne plus rien renvoyer.

En revanche, avec de la lumière, c'est l'inverse. Les énergies s'additionnent ! Plus tu mélanges de couleurs, plus tu vas vers le blanc. La synthèse additive, c'est ce que font les jeux de lumière au théâtre et au concert.

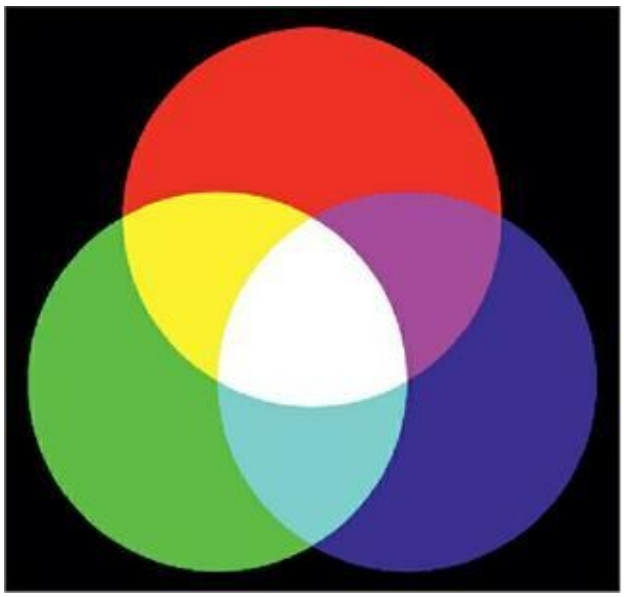


Figure 10.2 : Le spectre des couleurs visibles.

La lumière, ce sont des ondes électromagnétiques. Qui dit onde, dit fréquence. Le tableau suivant montre les fréquences des couleurs de l'arc-en-ciel. C'est avec ces valeurs que nous allons jouer dans ce chapitre.

Tableau 10.1 : Valeurs des couleurs principales

Couleur	*Longueur d'onde*	*Fréquence en térahertz*
Ultraviolet (UV)	moins de 380	plus de 850 THz
Violet	380–450 nm	668–789 THz
Bleu	450–495 nm	606–668 THz
Vert	495–570 nm	526–606 THz
Jaune	570–590 nm	508–526 THz
Orange	590–620 nm	484–508 THz
Rouge	620–750 nm	400–484 THz
Infrarouge	plus de 750	moins de 500 THz

Un térahertz vaut 1 000 gigahertz.

Ce qui est magique avec la lumière, c'est que les ondes s'additionnent. En musique, il existe aussi un bruit blanc, un peu comme lorsque tu enfonces toutes les touches du piano en même temps. C'est beaucoup moins transparent pour tes oreilles que la lumière du jour !

Par rapport au spectre des ondes lumineuses (Figure 10.3), à gauche, ce sont les fameux rayons ultraviolets qui nous bronzent et nous brûlent sur la plage. Encore plus à gauche arrivent les méchants rayons X puis, encore plus aigus, les très méchants rayons gamma de la radioactivité !

Du côté droit commence le domaine des infrarouges que nous connaissons bien maintenant. Encore plus grave en fréquence, nous arrivons en terrain connu : radars, micro-ondes, téléphones portables, radio… bref, encore des ondes. Heureusement qu'elles ne sont pas visibles, nous serions aveuglés !

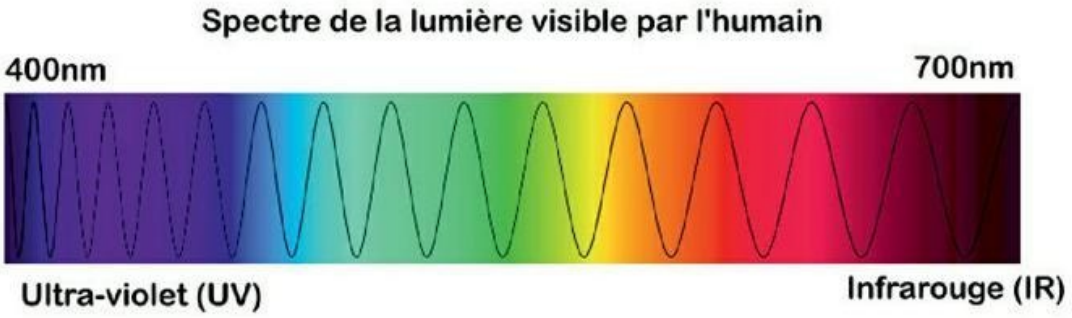


Figure 10.3 : Le spectre des fréquences de la lumière.

Mais que peut-on faire avec des diodes LED ? Il en existe de plusieurs couleurs, rouge, vert, jaune et bleu, et… c'est tout. Je ne compte pas les LED infrarouges, ni les LED ultraviolettes (cela existe).

La solution pour jouer avec les teintes est la diode tricolore.

La diode LED universelle

La LED tricolore, dite LED RGB (le G signifie Green), réunit dans une même capsule translucide une LED rouge, une verte et une bleue. Du fait que le boîtier n'est pas transparent, les trois sources de lumière se diffusent assez pour se mélanger.

Puisqu'il y a trois diodes LED, il nous faut six fils, deux pour chacune, n'est-ce pas ? Voyons… Si on réunissait les trois broches d'un côté sur une broche commune, cela fonctionnerait aussi bien, non ?

C'est bien le cas : la diode LED tricolore n'a que quatre broches. En conséquence, il y a deux variantes :

- soit on réunit les trois anodes (reliées au pôle plus), et c'est une diode à anode commune, AC (ou CA en anglais) ;
- soit on relie les trois cathodes (reliées à la masse GND), et c'est une diode à cathode commune, CC.

Figure 10.4 : Broches d'une LED RVB à anode (gauche) et à cathode commune.

Si tu récupères une LED RGB qui a perdu sa pochette, tu ne peux pas savoir si c'est une AC ou une CC. Heureusement, il suffit de tester. Si tu as un multimètre, règle-le sur « ohmmètre » . Dans un sens, le courant ne passe pas ; la résistance est infinie. Quand tu as trouvé le sens passant, repère la broche de la sonde rouge du multimètre : c'est l'anode. Si c'est la broche la plus longue, c'est une diode AC.

La logique de contrôle est inversée entre AC et CC :

- Avec une diode CC, on éteint la LED en amenant chaque anode au 0 V. Du fait que la cathode est reliée au 0 V en permanence, aucun courant ne passe plus. Pour obtenir du blanc, on porte les trois anodes au 5 V (1 logique). Du fait que la cathode est reliée au 0 V, le courant (limité par la résistance) passe.
- En revanche, avec une diode AC, on éteint la LED en amenant 5 V sur chaque cathode, donc au même potentiel que l'anode, et aucun courant ne passe.

CC semble plus naturel puisque l'intensité de lumière est proportionnelle à la valeur.

Sorties numériques

Avec un pilotage numérique par `digitalWrite()`, on peut donc obtenir les sept teintes de la synthèse additive (voir Figure 10.2) :

- les trois couleurs primaires rouge, vert et bleu ;
- les trois couleurs secondaires jaune (rouge+vert), rose (rouge + bleu) et bleu ciel (bleu + vert) ;
- et bien sûr le blanc avec les trois diodes allumées.

Il n'y a que sept teintes possibles en numérique.

Sorties analogiques PWM

Pour avoir plus de choix dans les couleurs, nous allons utiliser notre botte secrète : les sorties numériques qui simulent des sorties analogiques avec le mode PWM et la fonction `analogWrite()`.

En mode PWM, on dispose de 256 intensités possibles pour chaque couleur primaire (de 0 à 255). Pour notre premier programme, nous allons nous limiter aux deux valeurs extrêmes :

- 0 = éteinte ;
- 255 = lumière maximale.

Voyons d'abord comment monter le circuit dans les deux variantes possibles.

Montage AC (anode commune) de Tricolore

Les quatre fils nécessaires seront si possible ceux-ci :

- orange pour le rouge ;
- vert pour le vert ;
- blanc pour le bleu ;
- rouge pour le 5 V.

1. Implante la diode LED dans quatre demi-rangées de ta plaque d'essai (Figure 10.5).

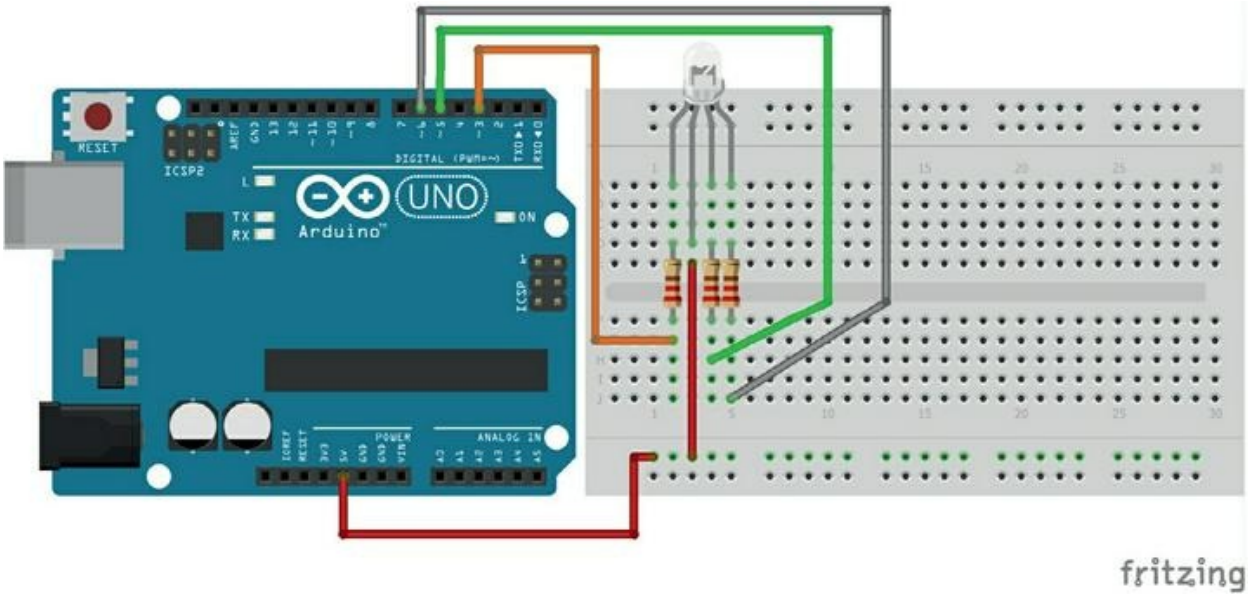


Figure 10.5 : Schéma de montage à anode commune.

Je conseille d'insérer la broche commune (anode, ici) en décalage d'un trou pour faciliter le repérage puisque tu ne vois plus les différences de longueur des pattes.

2. Implante une résistance de 270 ohms entre chacune des demi-rangées d'une des trois cathodes R, V, B et la demi-rangée d'en face.

3. Connecte chacune des trois demi-rangées où arrivent les résistances à trois broches numériques de la carte Arduino.

   La broche pour le rouge sur la sortie DIGITAL 3, la verte sur la sortie 5 et la bleue sur la sortie 6.

4. Connecte par un fil rouge l'anode commune au 5 V de la carte Arduino.

Le montage est terminé. Pour ceux qui ont une LED à cathode commune, voici le montage inversé.

Variante du montage à cathode commune (CC)

Voici les différences par rapport aux explications de la section précédente :

- Pour les 4 fils, le rouge du 5 V ne sert plus, mais il faut un noir pour la masse de la cathode commune.
- Les liaisons des trois couleurs sont les mêmes que pour une anode commune, sauf que ce sont trois anodes.
- La cathode commune doit bien sûr être reliée à la masse (une des trois sorties GND de la carte).

Codage de Tricolore AC

Pour plus de confort, nous allons définir deux fonctions locales. La première va décider de la luminosité des trois couleurs en attendant quatre paramètres :

- l'intensité pour le rouge, le vert et le bleu ;
- la durée de cet état d'allumage.

La seconde fonction locale se sert de la première pour tout éteindre.

Pour limiter le travail de reprise si tu as besoin de faire fonctionner le programme avec une diode LED à cathode commune, je définis six synonymes par des directives #define.

Dans cette première version, j'allume progressivement chacune des trois LED tour à tour. Nous obtenons ainsi les trois couleurs primaires en synthèse additive.

Listing 10.1 : Texte source de CH10_Tricolore1

```
// Tricolore1
// OEN170327

#define RMIN 255  // Si Anode Commune (AC)
#define VMIN 255
#define BMIN 255
#define RMAX 0
#define VMAX 0
#define BMAX 0

const byte yBroROUGE = 3;
const byte yBroVERT  = 5;
const byte yBroBLEU  = 6;

void setup() {
  allumerRVB(RMAX,VMAX,BMAX, 500); // Blanc
  allumerRVB(RMIN,VMIN,BMIN, 500); // Noir
}
```

```
void loop() {
  for (byte i = RMIN; i > RMAX; i--) {
    allumerRVB(i, VMIN, BMIN, 10);
  }
  eteindre(100);
  for (byte i = VMIN; i > VMAX; i--) {
    allumerRVB(RMIN, i, BMIN, 10);
  }
  eteindre(100);
  for (byte i = BMIN; i > BMAX; i--) {
    allumerRVB(RMIN, VMIN, i, 10);
  }
  eteindre(100);
}

void allumerRVB(byte rouge, byte vert, byte
bleu, int temps) {
  analogWrite(yBroROUGE, rouge);
  analogWrite(yBroVERT, vert);
  analogWrite(yBroBLEU, bleu);
  delay(temps);
}

void eteindre(int temps) {
  allumerRVB(RMIN, VMIN, BMIN, temps);
  delay(temps);
}
```

Variante à cathode commune

Grâce aux directives #define, nous modifions facilement le texte source pour une LED tricolore avec cathode commune (CC).

Pour une CC, il faut faire deux retouches :

1. Inverser les valeurs des six directives #define :

```
#define RMIN 0 // Si Cathode commune (CC)
#define VMIN 0
#define BMIN 0
#define RMAX 255
#define VMAX 255
#define BMAX 255
```

2. Inverser la logique des trois boucles de répétition (en rouge, les deux changements) :

```
for (byte i = RMIN; i < RMAX; i++) {
```

Cette retouche est à faire trois fois pour les trois LED.

Test du projet

Téléverse le projet et ouvre grands les yeux. Chaque LED doit briller de plus en plus puis s'éteindre.

Ajoutons les couleurs secondaires

Puisque la synthèse additive permet d'obtenir trois couleurs secondaires, enrichissons notre palette avec du jaune, du rose et du bleu ciel !

1. Enregistre le projet sous un nouveau nom, par exemple `Tricolore2`.

2. Positionne-toi juste avant l'accolade fermante de la fonction principale `loop()` et ajoute tout le bloc suivant. Ce sont trois autres blocs de répétition.

Listing 10.2 : Extrait du texte source de Tricolore2

```
// DEBUT DU PROGRAMME Tricolore1
// AJOUT COULEURS SECONDAIRE DANS loop()

  for (byte i = RMIN; i > RMAX; i--) {
    allumerRVB(i, i, BMIN, 10);
  }
  eteindre(100);
  for (byte i = VMIN; i > VMAX; i--) {
    allumerRVB(RMIN, i, i, 10);
  }
  eteindre(100);
  for (byte i = BMIN; i > BMAX; i--) {
    allumerRVB(i, VMIN, i, 10);
  }
  eteindre(100);
// SUITE DU PROGRAMME Tricolore1
```

Test du projet

C'est simple : téléverse et admire ! En éteignant la lumière, c'est encore plus joli. Puisque nous travaillons en pseudo-analogique, nous pouvons créer des nuances, beaucoup même, plus de 16 millions (256 x 256 x 256).

Mais calmons-nous. Nous allons nous contenter de combiner les teintes deux à deux :

- du rouge maximal, nous réduisons le rouge tout en augmentant le vert ;
- une fois le vert resplendissant, nous le diminuons tout en augmentant le bleu ;
- une fois le bleu bien brillant, nous le diminuons tout en augmentant le rouge.

Cela simule assez bien la palette de l'arc-en-ciel.

Tout est dans la nuance !

Puisque nous pouvons mélanger les couleurs, voyons comment passer par toutes les nuances de l'arc-en-ciel.

Tu devines qu'il va s'agir de commencer par une couleur que nous faisons monter, puis nous la réduisons tout en faisant monter la primaire suivante. L'astuce consiste à utiliser un tableau :

```
byte tyPrimaires[3];
```

Les trois cases du tableau mémorisent les valeurs d'intensité des trois LED. Au début de chaque tour de boucle, nous restockons la valeur minimale :

```
tyPrimaires[0] = RMIN;
tyPrimaires[1] = VMIN;
tyPrimaires[2] = BMIN;
```

Dans chaque tour, nous ne travaillons qu'avec deux des trois LED : la LED dont l'intensité est en train de diminuer et celle dont l'intensité va augmenter. Pour accéder aux bonnes cases du tableau, nous exploitons deux variables servant d'indices vers ces deux cases :

```
  byte yColDimi = 0;
  byte yColAugm;
```

Listing 10.3 : Texte source de Arkenciel

```
// Arkenciel
// OEN170327

#define RMIN 255  // Si anode commune
#define VMIN 255
#define BMIN 255
#define RMAX 0
#define VMAX 0
#define BMAX 0

const byte yBroROUGE = 3;
const byte yBroVERT  = 5;
const byte yBroBLEU  = 6;

void setup() {
  allumerRVB(RMAX,VMAX,BMAX, 200); // Pleins
feux puis
  allumerRVB(RMIN,VMIN,BMIN, 200); // Noir
}

void loop() {
  byte tyPrimaires[3];
  byte yColDimi = 0;
  byte yColAugm;

  tyPrimaires[0] = RMIN;
```

```
  tyPrimaires[1] = VMIN;
  tyPrimaires[2] = BMIN;

  for (byte i = RMIN; i > RMAX; i--) {
    allumerRVB(i, VMIN, BMIN, 10);
  }

  for (yColDimi = 0; yColDimi < 3; yColDimi
+= 1) {
    yColAugm = (yColDimi == 2 ? 0 : yColDimi
+ 1);    // NOUVEAU

    for(int i = 255; i > 0; i -= 1) {
      tyPrimaires[yColDimi] = (255-i); //
+=1 si AC
      tyPrimaires[yColAugm] = i;        // -
=1 si AC

      allumerRVB(tyPrimaires[0],
tyPrimaires[1], tyPrimaires[2], 10);
       delay(5);
    }
  }
  eteindre(1500);
}

void allumerRVB(byte rouge, byte vert, byte
bleu, int temps) {
  analogWrite(yBroROUGE, rouge);
  analogWrite(yBroVERT, vert);
  analogWrite(yBroBLEU, bleu);
  delay(temps);
```

```
}
void eteindre(int temps) {
  allumerRVB(RMIN, VMIN, BMIN, temps);
  delay(temps);
}
```

Pour faire d'abord monter l'intensité de la LED rouge seule, nous utilisons une boucle simple :

```
for (byte i = RMIN; i > RMAX; i--) {
  allumerRVB(i, VMIN, BMIN, 10);
}
```

La partie essentielle est la double boucle juste après. Dans le premier niveau, nous organisons trois tours de boucle en nous basant sur `yColDimi` qui progresse de 0 (LED rouge) à 2 (LED bleue).

```
for (yColDimi = 0; yColDimi < 3; yColDimi +=
1) {
```

Dans cette boucle, nous faisons un test qui utilise un nouvel opérateur : l'opérateur ternaire symbolisé par un signe point d'interrogation. Voici sa syntaxe générique :

```
( expression ? instruction_si_vraie :
instruction_si_fausse )
```

Voici comment nous l'utilisons :

```
yColAugm = (yColDimi == 2 ? 0 : yColDimi +
1);    // NOUVEAU
```

L'expression avant le point d'interrogation peut être soit vraie, soit fausse. Ici, nous testons si `yColDimi` vaut 2.

- Si c'est vrai, nous sélectionnons l'instruction juste après le signe. Ici, nous copions donc la valeur 0 dans la variable située tout à gauche (`yColAugm`).
- Si c'est faux (nous n'avons pas atteint la dernière des trois LED), nous exécutons l'instruction située après le signe deux-points. Ici, nous rendons `yColAugm` supérieure de 1 au numéro de LED de `yColDimi`. Cela désigne bien la LED suivante.

Souviens-toi des deux signes, d'abord le point d'interrogation, puis le deux-points. Tu remarques qu'il n'y a aucun point-virgule dans les instructions. C'est une écriture un peu spéciale, je l'avoue. Alors pourquoi ?

C'est simplement plus compact que le bloc suivant qui a exactement le même effet :

```
if (yColDimi == 2) {
   yColAugm = 0;
}
else {
  yColAugm = yColDimi +1;
}
```

Une fois ces choix de cases de tableaux réglés, il ne reste plus qu'à remplir avec des valeurs décroissantes pour la première des deux LED et croissantes pour l'autre :

```
for(int i = 255; i > 0; i -= 1) {
  tyPrimaires[yColDimi] = (255-i); // +=1 si
AC
  tyPrimaires[yColAugm] = i;         // -=1 si
AC
```

Si tu utilises une LED à cathode commune, il faut inverser les progressions comme suggéré dans les commentaires de fins de lignes.

Test du projet

Téléverse le projet pour admirer le passage par une foule de nuances.

Passons maintenant à un autre projet qui nous servira à piloter automatiquement la nuance affichée par la LED tricolore.

Le capteur de couleurs TCS 34725

Nous allons maintenant découvrir un nouveau type de capteur. C'est une sorte d'œil électronique qui sait voir les couleurs.

Le composant principal se trouve en plein milieu du circuit. C'est la puce juste à droite du pavé blanc et jaune qui est la LED servant à illuminer l'objet à analyser. La puce n'est pas opaque : tu peux voir la surface semi-conductrice. Cette surface est remplie de petites cellules sensibles à la lumière. Chaque cellule est dotée d'un filtre pour ne laisser passer que la lumière rouge, verte ou bleue.

Cette puce est accompagnée de quelques composants pour l'exploiter. La Figure 10.6 montre l'aspect général du module capteur.

À la différence du détecteur de mouvement PIR et du capteur de rayons infrarouges, le module ne se limite pas à trois broches de sortie. Il en a sept, parmi lesquelles bien sûr la broche du positif (marquée 3 V3, mais 5 V conviennent aussi) et celle du pôle négatif GND de l'alimentation. Tu vois les sept bornes dans le bas de la Figure 10.6.

Figure 10.6 : Le module capteur de couleurs.

Ce module est bien plus sophistiqué qu'un capteur simple. Il ne fonctionne pas en tout ou rien. Pour transmettre une valeur de couleur pour trois couleurs différentes, il faut envoyer plusieurs octets de données lorsqu'il est interrogé. Pourtant, parmi toutes les broches du module, une seule sert à renvoyer des données, la broche SDA. Comment est-ce possible ? Grâce à la norme I2C.

I2C pour communiquer

I2C signifie Inter-Integrated Circuit. C'est un jeu de règles (un protocole) qui définit comment doivent dialoguer deux équipements électroniques ou plus. Au départ, c'est la société Philips qui a inventé la norme TWI (Two Wire Interface) en 1982. Ce nom anglais indique que c'est une interface qui n'a besoin que de deux fils (wire). Comment ? Il suffit de brancher un fil de données sur la broche SDA (DA pour DAta) et un fil pour les tops d'une horloge sur la broche SCL (CL pour CLock).

C'est un protocole de dialogue en série, c'est-à-dire que les valeurs binaires sont transmises l'une après l'autre. Au début du chapitre, nous avons construit une sorte d'interface parallèle, puisqu'il y

avait un fil pour chacune des trois diodes LED. Pour les allumer toutes trois en même temps, nous avons mis les trois broches de sortie à l'état correspondant. C'est une commande en parallèle. Bien sûr, transmettre huit bits en parallèle va huit fois plus vite que les transmettre à la queue-leu-leu. De nos jours, les circuits sont devenus tellement performants que le mode série devient suffisant dans presque tous les domaines d'emploi.

Il existe un autre protocole de communication série beaucoup plus répandu et que tu utilises certainement : le protocole USB, le S signifiant « Série » . En USB 3.1, le débit va jusqu'à un milliard d'octets par seconde !

Dans la norme I2C, les bits de données sont envoyés l'un après l'autre au rythme du signal d'horloge. C'est un véritable dialogue qui est établi entre un maître (le mikon) et un esclave (le module capteur) : le module peut recevoir le code d'une commande, puis réagir en renvoyant le résultat de l'action déclenchée par cette commande dans le module.

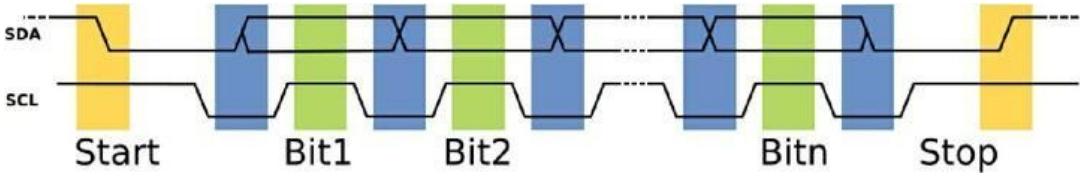


Figure 10.7 : Exemple de signal I2C.

Tu devines que ce riche dialogue fait de séquences Start, d'envoi de commandes, d'envoi de données et de séquences Stop réclame un grand nombre d'instructions. Pour simplifier l'utilisation du module, nous allons donc tirer avantage d'une librairie de fonctions déjà toute programmée. Mais procédons d'abord au montage du circuit.

Nomenclature de VolKool

Voici les composants à réunir :

- **Une plaque d'essai.** Tu peux réutiliser la plaque d'essai sur laquelle tu as implanté la diode LED tricolore. Je suppose que tu as assez de place à côté de cette diode.
- **Un capteur.** Pour le module capteur, il faut absolument disposer d'un modèle totalement compatible avec celui de la société Adafruit sous la référence TCS 34725. Nous avons en effet besoin de la librairie qui a été écrite par Adafruit pour ce module.

  Le problème qui peut se poser est que la société Adafruit (et ses revendeurs) propose ce module sans avoir soudé le connecteur de sortie sur le circuit. Le connecteur est livré, mais les soudures restent à faire et elles sont assez délicates car les contacts sont très proches les uns des autres. Il n'y a que deux solutions : soit tu connais quelqu'un qui peut te faire ces soudures, soit tu utilises la méthode de dépannage que je présente plus bas.
- **Des fils de connexion.** Tu auras besoin de quatre fils pour raccorder le module à la carte Arduino. Si tu as déjà installé un rail d'alimentation positif et un autre de masse GND sur la plaque, tu prendras l'alimentation sur les rails.
- **Une vis (option).** Si tu ne disposes pas d'une version du module avec le connecteur soudé, prévois une petite vis à bois avec laquelle tu pourras fixer le module sur une chute de bois par une de ses oreilles.

Montage de VolKool

Procédons maintenant au montage. Si possible, ne démonte pas le montage précédent avec sa LED tricolore. Nous allons le réutiliser.

1. Si le connecteur du module n'est pas soudé, fixe le module avec une vis sur une petite plaque de bois. Si le connecteur est soudé, insère le connecteur sur la plaque d'essai.
2. Relie par un fil vert ou bleu la broche SDA du module à la broche analogique **A4** sur la carte Arduino.
3. Relie avec un fil jaune ou blanc la broche SCL (horloge) à la broche analogique **A5** sur la carte Arduino.
4. Relie la broche de masse **GND** à la masse de la plaque ou de la carte.
5. Relie la broche du positif 3 V3 à la broche **5 V** du rail + de la carte. Pas d'inquiétude, le module comporte un régulateur et accepte aussi 5 V.
6. Si tu n'as pas de connecteur soudé, il faut sacrifier une extrémité de chacun des quatre fils en coupant un connecteur, puis en dénudant le fil sur 2 cm et en le torsadant. Tu fais ensuite passer chaque fil dans le trou de la broche correspondante, puis tu reboucles en torsade. Fixe le tout avec un point de colle. Ce n'est pas une solution élégante, mais si tu ne tires plus sur le module, cela devrait tenir.

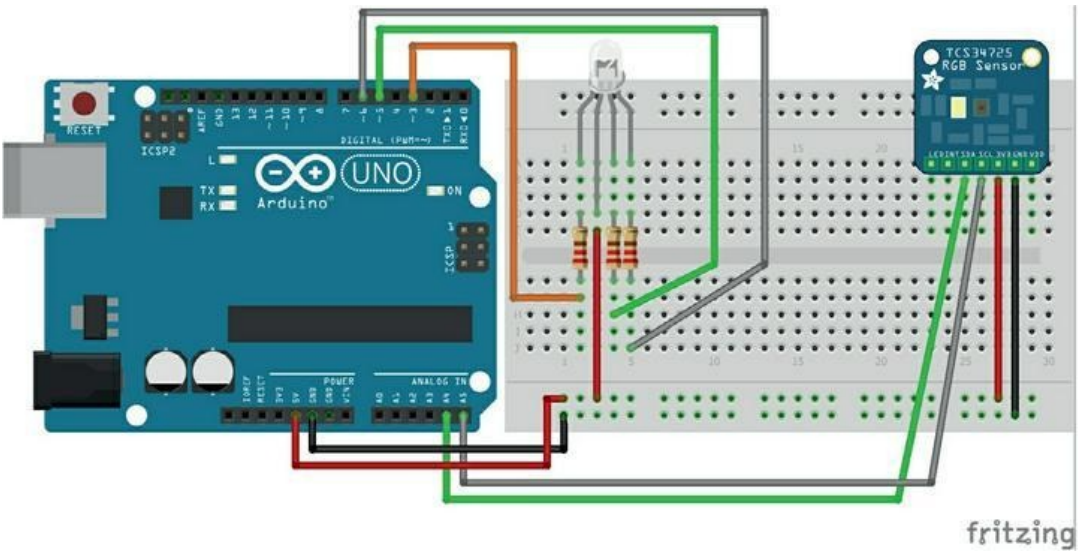


Figure 10.8 : Montage de VolKool.

Le montage est terminé, mais nous ne pouvons pas pour autant nous lancer dans un premier test. Il faut d'abord installer la librairie appropriée.

Installons la librairie du module

La librairie offerte par la société Adafruit est reconnue par l'atelier IDE Arduino comme faisant partie des plus demandées. Tu peux donc procéder sans crainte à son installation avec le module de gestion des librairies.

1. Démarre l'atelier Arduino si nécessaire.
2. Ouvre le menu **Croquis** et choisis **Inclure une bibliothèque** (librairie).
3. Choisis la commande **Gérer les bibliothèques**.

   Tu vois apparaître une boîte présentant la liste des librairies déjà installées et celles qui peuvent l'être.

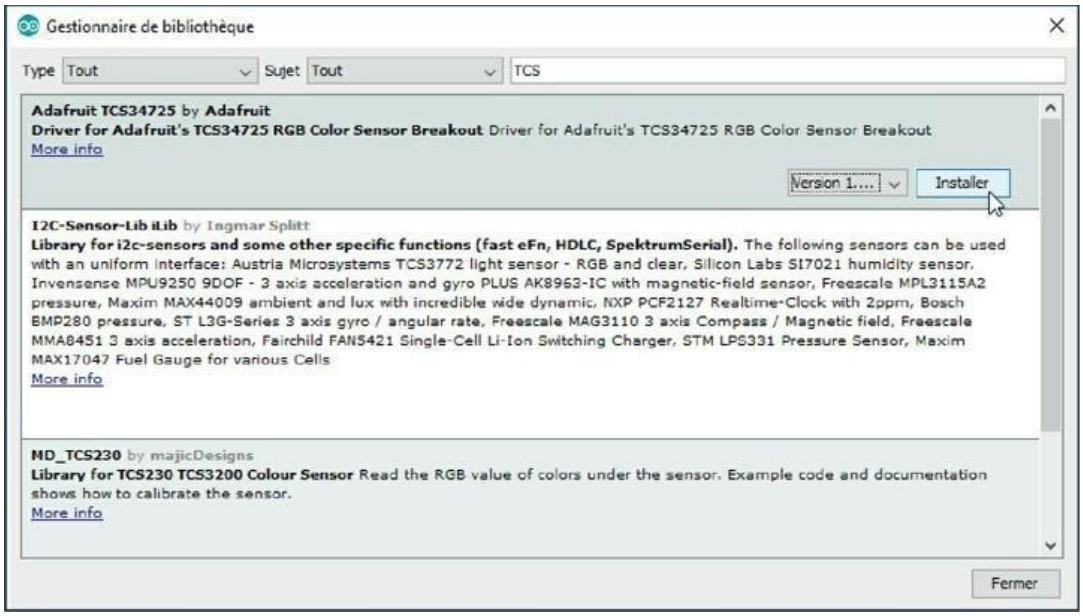

Gestionnaire de bibliothèque
Type Tout
Sujet Tout
TCS
Adafruit TCS34725 by Adafruit
Driver for Adafruit's TCS34725 RGB Color Sensor Breakout Driver for Adafruit's TCS34725 RGB Color Sensor Breakout
More info
Version 1....
Installer
I2C-Sensor-Lib iLib by Ingmar Splitt
Library for i2c-sensors and some other specific functions (fast eFn, HDLC, SpektrumSerial). The following sensors can be used with an uniform interface: Austria Microsystems TCS3772 light sensor - RGB and clear, Silicon Labs SI7021 humidity sensor, Invensense MPU9250 9DOF - 3 axis acceleration and gyro PLUS AK8963-IC with magnetic-field sensor, Freescale MPL3115A2 pressure, Maxim MAX44009 ambient and lux with incredible wide dynamic, NXP PCF2127 Realtime-Clock with 2ppm, Bosch BMP280 pressure, ST L3G-Series 3 axis gyro / angular rate, Freescale MAG3110 3 axis Compass / Magnetic field, Freescale MMA8451 3 axis acceleration, Fairchild FAN5421 Single-Cell Li-Ion Switching Charger, STM LPS331 Pressure Sensor, Maxim MAX17047 Fuel Gauge for various Cells
More info
MD_TCS230 by majicDesigns
Library for TCS230 TCS3200 Colour Sensor Read the RGB value of colors under the sensor. Example code and documentation shows how to calibrate the sensor.
More info
Fermer

Figure 10.9 : Installer une librairie.

4. Dans la zone de saisie en haut à droite, saisis les trois lettres `TCS`. Cela suffit à filtrer l'affichage.

   Normalement, la première librairie de la liste est celle dont nous avons besoin, `Adafruit TCS34725`.

5. Clique dans la surface de la librairie. Tu vois apparaître à droite un bouton **Installer**. Clique-le.

   Patiente un peu. La librairie va être installée avec ses exemples.

6. Quitte l'atelier et redémarre.

Chargeons le programme de test

Pour le programme de test, tu peux partir de celui qui a été installé d'office avec la librairie : TCS34725.

1. Dans l'atelier, ouvre le menu **Fichier** puis le sous-menu **Exemples**. Sélectionne **Gérer les bibliothèques**.
2. Descends dans la liste jusqu'au nom de la librairie que tu viens d'installer, `Adafruit TCS34725`. Accède au sous-menu (Figure 10.10) et sélectionne l'exemple `TCS34725`.

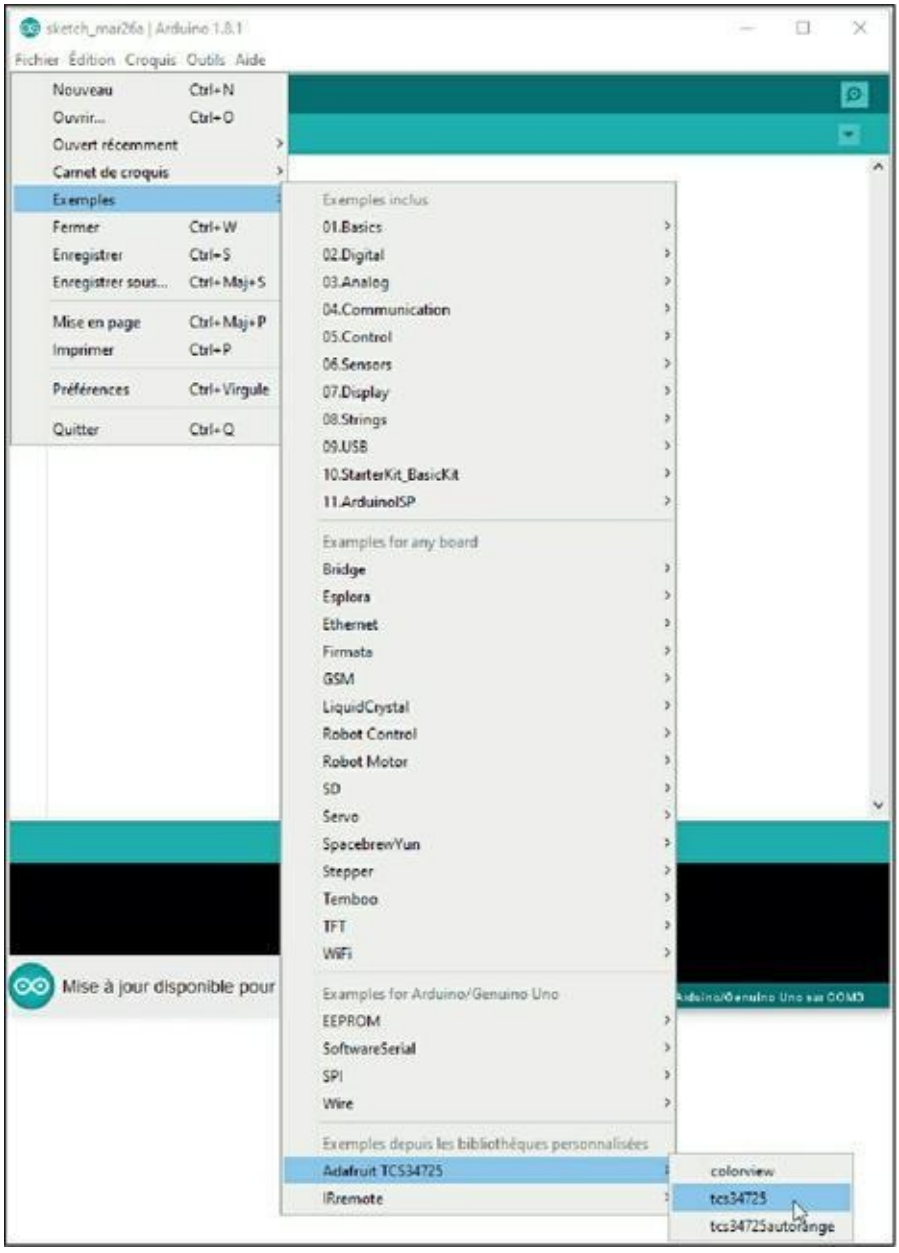


Figure 10.10 : Chargement du programme de test TCS34725.

3. Sauvegarde le programme sous le nom `VolKool1`.

Tu peux aussi saisir l'exemple suivant qui en est inspiré. La différence est que j'ai aménagé l'exemple en francisant les noms de variables et de fonctions.

Cet exemple fait partie de ceux fournis avec le livre sur le site sous le nom `CH10C_VolKool1`.

Listing 10.4 : Texte source de VolKool1

```
// VolKool1
// OEN170327

#include <Wire.h>
#include "Adafruit_TCS34725.h"

/* Exemple de la librairie Adafruit TCS34725
*/
/* SCL sur broche A5, SDA sur A4, VDD sur
3.3V et GND sur GND */

// Une seule ligne ici:
Adafruit_TCS34725 oVolKool =
Adafruit_TCS34725(TCS34725_
INTEGRATIONTIME_700MS, TCS34725_GAIN_1X);

void setup(void) {
  Serial.begin(9600);
  if (oVolKool.begin()) { Serial.println(
F("Capteur actif !") ); }
```

```
  else {
    Serial.println( F("TCS34725 absent. Voir
les connexions!") );
    while (1); }

}

void loop(void) {
  uint16_t iR, iV, iB, iC, iTempera, lux;

  oVolKool.getRawData(&iR, &iV, &iB, &iC);
  iTempera =
oVolKool.calculateColorTemperature(iR, iV,
iB);
  lux = oVolKool.calculateLux(iR, iV, iB);

  Serial.print("Temp. de couleur: ");
Serial.print(iTempera, DEC);
  Serial.print(" K - ");
  Serial.print("Lux: "); Serial.print(lux,
DEC);
  Serial.println(" - ");
  Serial.print("Rouge: "); Serial.print(iR,
DEC); Serial.print(" ");
  Serial.print("Vert:   "); Serial.print(iV,
DEC); Serial.print(" ");
  Serial.print("Bleu:   "); Serial.print(iB,
DEC); Serial.print(" ");
  Serial.print("C: "); Serial.print(iC,
DEC); Serial.print(" ");
  Serial.println();
  Serial.println(" ");
```

```
}
```

Ce programme est inspiré de celui proposé par Adafruit sous le nom TCS34725.

En début de programme, nous demandons d'utiliser le contenu de deux librairies de fonctions :

```
#include <Wire.h>
#include "Adafruit_TCS34725.h"
```

La seconde librairie est celle que nous venons d'installer. La première, `Wire.h`, est installée d'office. Elle contient les fonctions pour exploiter le protocole I2C qui régit le dialogue entre le mikon et le capteur.

Nous commençons par créer un objet `oVolKool` à partir de la classe nommée `Adafruit_TCS34725`. Cette création est réalisée en appelant une méthode avec deux paramètres. L'instruction est si longue qu'elle ne tient pas sur une ligne de ce livre, mais c'est bien une seule ligne :

```
// Une seule ligne ici:
Adafruit_TCS34725 oVolKool =
Adafruit_TCS34725(TCS34725_
INTEGRATIONTIME_700MS, TCS34725_GAIN_1X);
```

Dans la fonction de préparation, nous vérifions que le capteur est prêt avec

un appel à `oVolKool.begin()`.

Dans la boucle principale, tu remarques un étrange nom de type de données :

```
uint16_t iR, iV, iB, iC, iTempera, lux;
```

Le mot clé uint16_t est bien le nom d'un type de données. C'est une version abrégée de **unsigned int**, mais obligatoirement sur deux octets (16 bits). Cela rend le programme plus portable vers un autre mikon. En effet, celui de la carte Uno est sur 16 bits, mais d'autres processeurs fonctionnent avec une largeur de 32 bits pour les entiers, d'autres sur 8 bits seulement. Avec ce type exact, tu obliges le compilateur à utiliser deux octets pour les variables concernées.

Nous enchaînons ensuite avec trois appels à des fonctions de la librairie pour peupler nos six variables :

```
  oVolKool.getRawData(&iR, &iV, &iB, &iC);
  iTempera =
oVolKool.calculateColorTemperature(iR, iV,
iB);
  lux = oVolKool.calculateLux(iR, iV, iB);
```

- La première, `getRawData()`, rapatrie les valeurs des trois composantes de couleurs et la luminosité globale.
- La deuxième, `calculateColorTemperature()`, récupère la température de couleur.
- La dernière, `calculateLux()`, obtient l'intensité lumineuse exprimée dans l'unité standard, le lux.

La fin du programme n'est qu'une suite d'affichages.

Il n'est pas nécessaire d'ajouter un appel à `delay()` pour ralentir les affichages. Le mikon est suffisamment occupé avec tous ces appels.

Testons le capteur de couleurs

Réunis quelques objets ayant une surface unie d'une couleur bien marquée. Des fruits, des surligneurs, des jetons de jeu de société, etc.

1. Téléverse le projet et ouvre la fenêtre du Moniteur série (Figure 10.11).

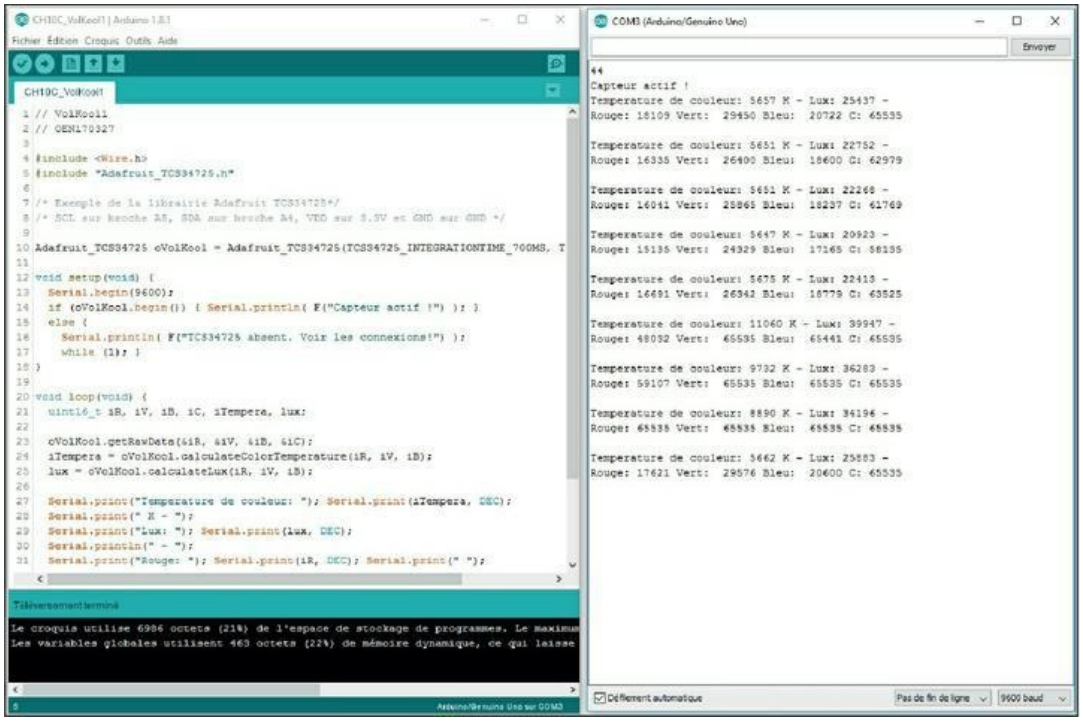

CH10C_VolKool1 | Arduino 1.8.1
Fichier Édition Croquis Outils Aide
CH10C_VolKool1
// VolKool1
// OEN170327
#include <Wire.h>
#include "Adafruit_TCS34725.h"
/* Exemple de la librairie Adafruit TCS34725*/
/* SCL sur broche A5, SDA sur broche A4, VDD sur 3.3V et GND sur GND */
Adafruit_TCS34725 oVolKool = Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_700MS, T
void setup(void) {
Serial.begin(9600);
if (oVolKool.begin()) { Serial.println(F("Capteur actif !")); }
else {
Serial.println(F("TCS34725 absent. Voir les connexions!"));
while (1); }
}
void loop(void) {
uint16_t iR, iV, iB, iC, iTempera, lux;
oVolKool.getRawData(&iR, &iV, &iB, &iC);
iTempera = oVolKool.calculateColorTemperature(iR, iV, iB);
lux = oVolKool.calculateLux(iR, iV, iB);
Serial.print("Temperature de couleur: "); Serial.print(iTempera, DEC);
Serial.print(" K - ");
Serial.print("Lux: "); Serial.print(lux, DEC);
Serial.print(" - ");
Serial.print("Rouge: "); Serial.print(iR, DEC); Serial.print(" ");
Le croquis utilise 6986 octets (21%) de l'espace de stockage de programmes. Le maximu
Les variables globales utilisent 463 octets (22%) de mémoire dynamique, ce qui laisse
Arduino/Genuino Uno sur COM3
COM3 (Arduino/Genuino Uno)
Envoyer
Capteur actif !
Temperature de couleur: 5657 K - Lux: 25437 -
Rouge: 18109 Vert: 29450 Bleu: 20722 C: 65535
Temperature de couleur: 5651 K - Lux: 22752 -
Rouge: 16335 Vert: 26400 Bleu: 18600 C: 62979
Temperature de couleur: 5651 K - Lux: 22268 -
Rouge: 16041 Vert: 25865 Bleu: 18237 C: 61769
Temperature de couleur: 5647 K - Lux: 20923 -
Rouge: 15135 Vert: 24329 Bleu: 17165 C: 58135
Temperature de couleur: 5675 K - Lux: 22413 -
Rouge: 16491 Vert: 26342 Bleu: 18779 C: 63525
Temperature de couleur: 11060 K - Lux: 39947 -
Rouge: 48032 Vert: 65535 Bleu: 65441 C: 65535
Temperature de couleur: 9732 K - Lux: 36283 -
Rouge: 59107 Vert: 65535 Bleu: 65535 C: 65535
Temperature de couleur: 8890 K - Lux: 34196 -
Rouge: 65535 Vert: 65535 Bleu: 65535 C: 65535
Temperature de couleur: 5662 K - Lux: 25883 -
Rouge: 17621 Vert: 29576 Bleu: 20600 C: 65535
Défilement automatique
Pas de fin de ligne
9600 baud

Figure 10.11 : Affichage des composantes de couleurs.

2. Approche un premier objet du capteur. Le capteur est myope. Il faut vraiment aller presque au contact.

3. Lis et compare les valeurs. Selon la couleur de l'objet, une des composantes va augmenter de valeur bien plus que les deux autres.

Normalement, tu dois constater que les valeurs des couleurs varient correctement en fonction de la couleur de l'objet présenté. Nous

sommes prêts à donner naissance au voleur de couleurs !

Copier, c'est beau (VolKool2)

Pour terminer le projet, nous combinons le capteur de couleurs à la diode LED RGB des premiers projets du chapitre.

Si tu avais entre-temps démonté la partie Tricolore du circuit, revois la procédure autour de la Figure 10.5 plus haut.

Le projet va devenir un circuit fermé :

- Nous captons en entrée trois valeurs de couleurs.
- Nous les traitons dans le mikon en appliquant une correction de gamme.
- Nous décidons en sortie de la teinte de la LED tricolore.

Le texte source ne comporte qu'une vraie nouveauté : la table des gamma.

Listing 10.5 : Texte source de VolKool2

```
// VolKool2
// OEN170327

#include <Wire.h>
#include "Adafruit_TCS34725.h"

// Résistance de 220 ohm sur les 3 LED.
// Si la verte brille trop, double la valeur
de sa résistance
#define brRouge 3
```

```
#define brVerte 5
#define brBleue 6

#define anodeComm true  // Indiquer false si
cathode comm.

// Table des gamma
byte tabGamma[256];

Adafruit_TCS34725 monTCS =

Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_50MS
 TCS34725_GAIN_4X);

void setup() {
  Serial.begin(9600);
  Serial.println("Le voleur de couleurs !");

  if (monTCS.begin()) {
    Serial.println("Capteur actif.");
  } else {
    Serial.println("Pas de TCS34725...
verifier les branchements.");
    while (1);
  }

  pinMode(brRouge, OUTPUT);
  pinMode(brVerte, OUTPUT);
  pinMode(brBleue, OUTPUT);

  // Merci PhilB pour la table gamma.
  // Elle rééquilibre la luminosité selon
```

```
l'œil humain.
  for (int i=0; i<256; i++) {
     float x = i;
    x /= 255;
    x = pow(x, 2.5);
    x *= 255;

    if (anodeComm) {
      tabGamma[i] = 255 - x;
    } else {
      tabGamma[i] = x;
    }
    //Serial.println(tabGamma[i]);
  }
}

void loop() {
  uint16_t clear, iRO, iVE, iBL;

  monTCS.setInterrupt(false);
  delay(60);                        // Patiente
50ms pour la lecture
  monTCS.getRawData(&iRO, &iVE, &iBL,
&clear);
  monTCS.setInterrupt(true);

  Serial.println();
  Serial.print("C:\t"); Serial.print(clear);
  Serial.print("\tR:\t"); Serial.print(iRO);
  Serial.print("\tG:\t"); Serial.print(iVE);
  Serial.print("\tB:\t");
Serial.println(iBL);
```

```
  // Calcul des valeurs hexa pour le Web
  uint32_t sum = clear;
  float r, g, b;
  r = iRO;  r /= sum;
  g = iVE;  g /= sum;
  b = iBL;  b /= sum;
  r *= 256; g *= 256; b *= 256;

  Serial.print(" Couleurs web \t\t");
  Serial.print((int)r, HEX);
  Serial.print((int)g, HEX);
  Serial.print((int)b, HEX);
  Serial.println();

  Serial.print(" Couleurs de base \t");
  Serial.print((int)r ); Serial.print(" ");
  Serial.print((int)g ); Serial.print(" ");
  Serial.println((int)b );

  analogWrite(brRouge, tabGamma[(int)r]);
  analogWrite(brVerte, tabGamma[(int)g]);
  analogWrite(brBleue, tabGamma[(int)b]);
  delay(1000);
}
```

La table des gamma

Sans la correction de gamma, certaines teintes vont sembler beaucoup plus lumineuses que les autres. L'œil humain est entraîné depuis des milliers d'années à repérer sa nourriture dans le décor

en plein jour, mais la nuit, il doit bien voir arriver un prédateur qui viendrait troubler son sommeil. Le résultat est que les teintes moyennes telles que les diodes LED les présentent sont trop lumineuses.

La correction gamma consiste à réduire la luminosité de certaines valeurs plus que d'autres.

Le bloc qui s'en charge dans l'exemple utilise une variable non entière : elle est du type float. Ce type permet de stocker une valeur avec des chiffres derrière la virgule.

```
for (int i=0; i<256; i++) {
   float fGam = i;
   fGam /= 255;
  fGam = pow(fGam, 2.5);
   fGam *= 255;

  if (anodeComm) {
     tabGamma[i] = 255 - fGam;
  } else {
     tabGamma[i] = fGam;
  }
  //Serial.println(tabGamma[i]);
}
```

Ici, nous stockons 256 valeurs de correction dans un tableau.

Tu peux voir qu'une valeur non entière est écrite avec un point en langage Arduino, comme dans la quatrième ligne du bloc pour le facteur de correction de 2,5 :

```
fGam = pow(fGam, 2.5);
```

Dans les calculs, il y a une division et une multiplication, toutes deux écrites sous forme abrégée :

```
fGam /= 255;                // Comme fGam = fGam
/ 255;
fGam = pow(fGam, 2.5);
fGam *= 255;                // Comme fGam = fGam
* 255;
```

Nous utilisons aussi la fonction `pow()`, qui fait un calcul de puissance.

Test du projet

Connecte la carte et téléverse.

Une fois le programme démarré, présente un objet coloré devant le capteur.

Le Moniteur série te donne une information utile : la valeur hexadécimale des trois composantes, telle que tu peux l'indiquer dans tout programme fonctionnant sur le Web :

- dans les pages HTML ;
- dans les feuilles de styles CSS ;
- dans les instructions JavaScript.

Par exemple, `45A04C` correspond à un vert moyen.

La notation hexadécimale va de seize en seize et non de dix en dix. Les lettres A à F symbolisent les chiffres hexa 10 à 15. La valeur d'exemple 45A04C se divise en trois couples :

- 45 pour le rouge, soit 4 seizaines + 5 = 69 ;
- A0 pour le vert, soit 10 seizaines + 0 = 160 ;
- 4C pour le bleu, soit 4 seizaines + 12 = 76.

Tu peux tester et trouver toutes les nuances sur un des nombreux sites d'aide à la conception Web, comme http://html-color-codes.info.

Pour vérifier l'effet de la correction de gamma, teste à nouveau le projet Arkenciel : les teintes ne sont pas aussi bien équilibrées.

Observe la LED tricolore : elle devrait prendre quasiment la même teinte que l'objet.

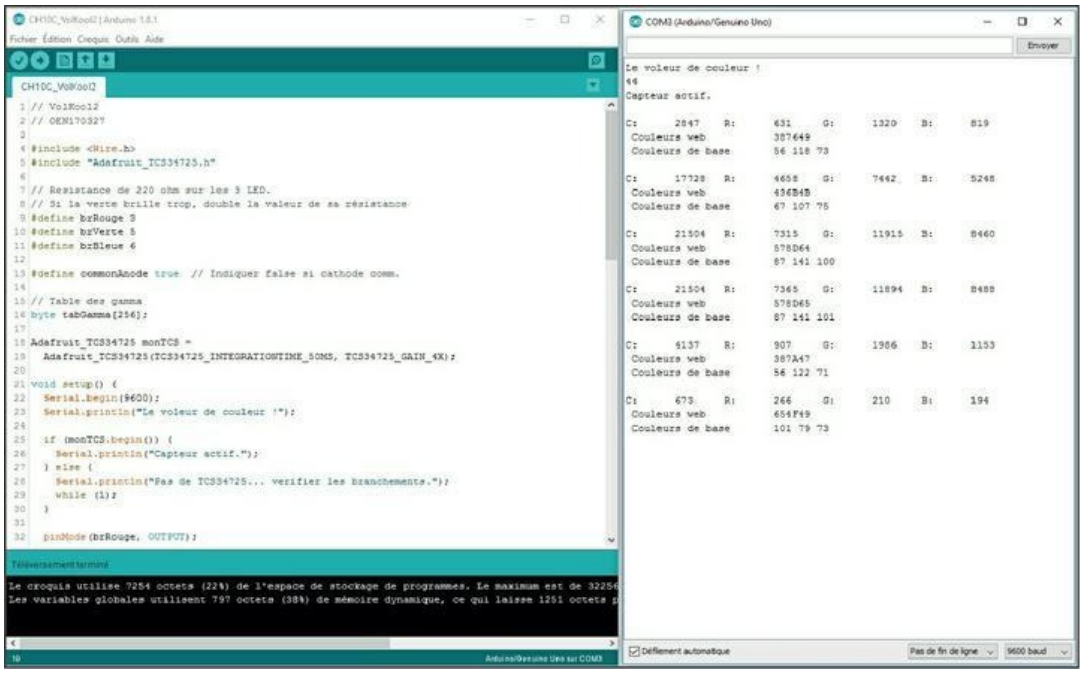

CH10C_VolKool2
// VolKool2
// OEN170327
#include <Wire.h>
#include "Adafruit_TCS34725.h"
// Resistance de 220 ohm sur les 3 LED.
// Si la verte brille trop, double la valeur de sa résistance
#define brRouge 3
#define brVerte 5
#define brBleue 6
#define commonAnode true // Indiquer false si cathode comm.
// Table des gamma
byte tabGamma[256];
Adafruit_TCS34725 monTCS =
Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_50MS, TCS34725_GAIN_4X);
void setup() {
Serial.begin(9600);
Serial.println("Le voleur de couleur !");
if (monTCS.begin()) {
Serial.println("Capteur actif.");
} else {
Serial.println("Pas de TCS34725... verifier les branchements.");
while (1);
}
pinMode(brRouge, OUTPUT);
Le croquis utilise 7254 octets (22%) de l'espace de stockage de programmes. Le maximum est de 32256
Les variables globales utilisent 797 octets (38%) de mémoire dynamique, ce qui laisse 1251 octets p
COM3 (Arduino/Genuino Uno)
Envoyer
Le voleur de couleur !
44
Capteur actif.
C: 2847 R: 631 G: 1320 B: 819
Couleurs web 387649
Couleurs de base 56 118 73
C: 17728 R: 4658 G: 7442 B: 5248
Couleurs web 436B4B
Couleurs de base 67 107 75
C: 21504 R: 7315 G: 11915 B: 8460
Couleurs web 578D64
Couleurs de base 87 141 100
C: 21504 R: 7365 G: 11894 B: 8488
Couleurs web 578D65
Couleurs de base 87 141 101
C: 6137 R: 907 G: 1986 B: 1153
Couleurs web 387A47
Couleurs de base 56 122 71
C: 673 R: 266 G: 210 B: 194
Couleurs web 654F49
Couleurs de base 101 79 73
Défilement automatique
Pas de fin de ligne
9600 baud

Figure 10.12 : Le Moniteur série de VolKool2.

Figure 10.13 : Test de VolKool2.

Félicitations ! Tu maîtrises maintenant les sons et les lumières avec ta carte Arduino.

Récapitulons

Dans ce chapitre, nous avons vu :

- les ondes de lumière et comment générer des nuances ;
- la diode LED RGB ;
- un capteur de couleurs et sa librairie.

Semaine 4 : Le temps des lettres

Au menu cette semaine :

Chapitre 11 : Le temps liquide

Chapitre 12 : Un peu d'échologie

Chapitre 11
Le temps liquide

AU MENU DE CE CHAPITRE :

- **Des afficheurs avec du caractère**
- **L'alphabet du mikon**
- **Prends ton temps**

Dans les précédents chapitres, nous avons appris au mikon à communiquer avec le monde des humains en envoyant des données par le câble USB vers l'ordinateur, ce qui a permis d'afficher du texte et des valeurs numériques dans la fenêtre du Moniteur série. Ce canal de communication restera toujours indispensable pour réaliser la mise au point des projets.

Mais ta carte Arduino commence à en avoir assez d'être retenue par un fil à la patte. Comment peut-on faire afficher un message ou des valeurs en n'alimentant la carte Arduino qu'avec une pile, c'est-à-dire en débranchant enfin la carte de la liaison USB ?

Bienvenue dans le vaste monde des afficheurs. Du plus simple au plus sophistiqué, tu disposes de toute une palette de composants permettant d'afficher du texte ou des graphiques. Faisons d'abord le tour des différents types d'afficheurs disponibles. Je te proposerai ensuite de choisir un type en particulier, pour que le montage reste simple et que la programmation ne soit pas trop complexe.

Les sections suivantes présentent les grandes catégories d'afficheurs disponibles.

Matrice de LED

Il suffit de réunir plusieurs diodes LED miniatures sous forme de lignes et de colonnes, et de noyer le tout dans un boîtier carré. Les plus répandus de ces composants offrent 8 lignes sur 8 colonnes, soit 64 LED.

Tu devines que pour connecter ce genre de matrice à une carte Arduino, il va falloir un grand nombre de fils. Il ne te restera ensuite plus beaucoup de broches disponibles pour faire autre chose.

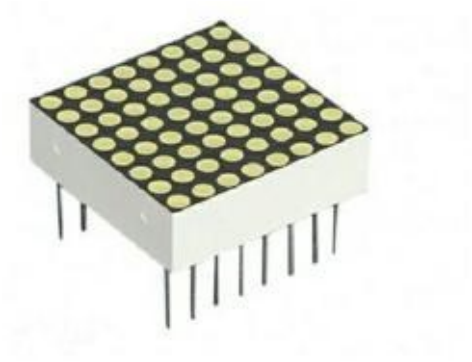


Figure 11.1 : Matrice de LEDS 8x8.

Afficheur sept segments

C'est le plus ancien composant d'affichage. Le principe est de déformer les LED pour qu'elles aient non plus la forme d'un point, mais celle d'un trait nommé segment (Figure 11.2).

Figure 11.2 : Vue générale d'un afficheur LED sept segments.

On peut ainsi afficher tous les chiffres ainsi qu'un certain nombre de lettres, mais pas toutes. En effet, avec sept segments seulement, les traits obliques ne sont pas possibles (Figure 11.3).

Figure 11.3 : Quelques dessins de chiffres dans un afficheur sept segments.

Pour afficher quatre chiffres, ce qui est souvent le cas pour les codes d'entrée des immeubles, il faut donc quatre afficheurs à ce segment, ce qui suppose en théorie 4 fois 8 connexions, soit 32 fils à brancher. Halte-là ! Tu te souviens que la carte Arduino et donc le mikon n'offrent que 14 sorties numériques. On est loin du compte.

Des parallèles en série

L'être humain est ingénieux. Il cherche toujours une solution lorsque c'est possible. Pour piloter quatre afficheurs sept segments lorsque l'on n'a pas assez de broches de sortie, on utilise un circuit spécial appelé *registre à décalage* (*shift register*). Ce composant reçoit les données l'une après l'autre sur une seule entrée, bit après bit, et les stocke dans ses petites cases de mémoire (les registres) en décalant d'une case après chaque bit.

Il possède huit sorties en parallèle, un peu comme lorsque les chevaux de course viennent en file indienne se placer côte à côte dans leurs boxes sur la ligne de départ. Lorsque l'on envoie une

commande spéciale (le signal du départ), l'état des huit sorties est rendu accessible en même temps à ce qui y est branché, par exemple l'afficheur. Avec cette technique, il suffit de quatre fils de données pour piloter quatre afficheurs à sept segments. Je n'en dirai pas plus dans ce livre, car des afficheurs plus polyvalents sont dorénavant disponibles à bas coût.

Afficheurs LCD

La troisième et dernière catégorie est celle des afficheurs à cristaux liquides ou LCD (liquid crystal display). Il y a deux sous-catégories :

Écrans LCD alphanumériques

Un afficheur LCD alphanumérique ne peut afficher que des caractères de l'alphabet et quelques signes de ponctuation. Les afficheurs les plus courants offrent 16 caractères affichables par ligne et 2 lignes. Leur référence comprend souvent la mention « 1602 » ; tu devines pourquoi.

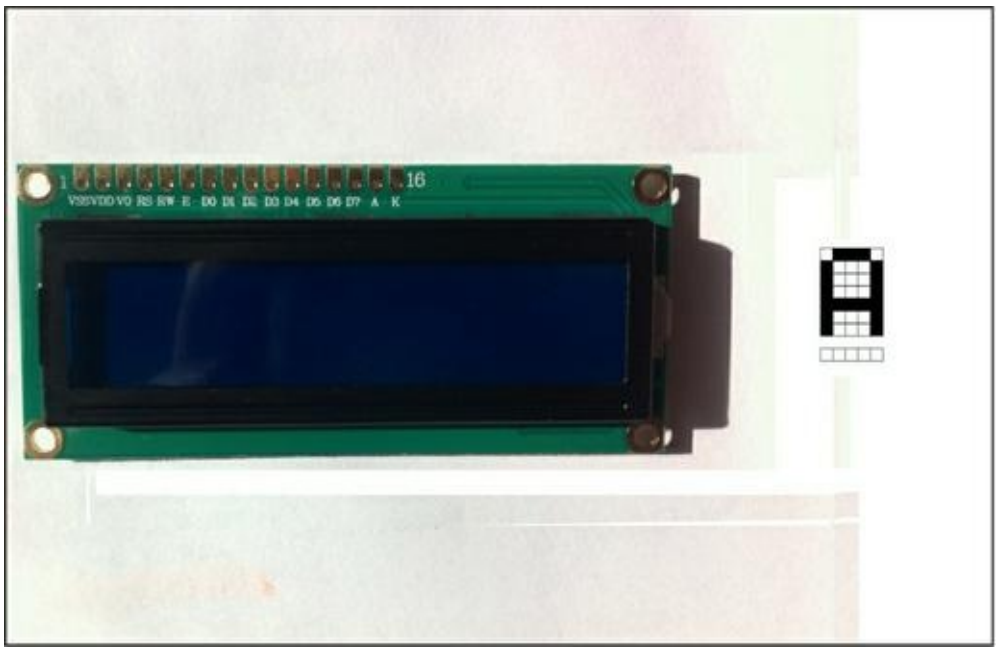


Figure 11.4 : Afficheur LCD alphanumérique.

Afficheurs LCD graphiques

Les afficheurs LCD graphiques ont des écrans composés d'une très grande quantité de points (les pixels). Les téléviseurs et les écrans d'ordinateur sont des afficheurs LCD graphiques, et les telliphones en ont tous un, souvent tactile.

Figure 11.5 : Afficheur LCD graphique couleur.

Je ne vais utiliser dans ce chapitre que le modèle le plus simple, qui est l'afficheur LCD alphanumérique, ou afficheur LCD en mode texte. Les afficheurs graphiques coûtent beaucoup plus cher et demandent beaucoup plus de travail au processeur.

Un écran LCD en mode texte demande beaucoup moins de traitement au processeur parce qu'il permet uniquement d'afficher un caractère parmi tous ceux qui sont prédessinés dans sa mémoire locale. Ce sont les caractères ASCII. Voyons cela en détail.

Toujours avoir bon caractère

Dès les débuts de l'informatique, il a été nécessaire de se mettre d'accord sur une série de codes numériques en correspondance avec des dessins de lettres de l'alphabet. Les Américains étant déjà à ce moment en avance dans la science informatique, c'est un standard américain qui a été choisi pour la correspondance entre code et dessins des lettres : le codage et la table ASCII.

Cet acronyme ASCII (prononce « aski ») signifie American Standard Code for Information Interchange. En bon français, c'est un standard américain pour échanger des informations, sous-entendu entre un ordinateur et un être humain, mais également entre deux ordinateurs.

Pour afficher et imprimer facilement du texte, la solution la plus efficace consiste à stocker directement dans l'écran, plutôt la carte vidéo, ou dans l'imprimante tous les dessins des caractères (ce qu'on appelle des glyphes). Il ne reste ensuite plus qu'à attribuer un code numérique à chaque dessin.

Dans la table ASCII, les lettres de l'alphabet anglais de A à Z (en lettres majuscules) correspondent aux codes numériques suivants :

Tableau 11.1 : Codes ASCII des lettres en majuscules

Valeur décimale	*Dessin*
65	A
66	B
67	C
etc.	
90	Z

Les chiffres de 0 à 9 sont codés ainsi :

Tableau 11.2 : Codes ASCII des chiffres

Valeur décimale	*Dessin*
48	0
49	1
50	2
51	3
52	4
53	5
54	6
55	7
56	8
57	9

Grâce à cette convention, pour afficher ou imprimer la lettre A majuscule, il suffit d'envoyer le code 65 à l'écran ou à l'imprimante.

Et pour afficher la valeur numérique 65, il faut envoyer l'octet 65 ? Pas du tout ! Cherche dans le Tableau 11.2 ! Pour afficher les deux chiffres 6 et 5, il faut envoyer les deux codes 54 et 53. Il ne faut pas confondre les valeurs numériques (le mikon ne sait rien traiter d'autre) et les dessins des chiffres.

En plus des 26 lettres de l'alphabet en majuscules et des 10 chiffres, la table ASCII contient l'alphabet en lettres minuscules, des signes de ponctuation ainsi qu'un certain nombre de caractères qui ne

peuvent être ni affichés, ni imprimés. Ce sont des caractères de contrôle ou d'échappement. Ce sont tous les codes inférieurs à 32.

C'est parmi ces codes que se trouve celui pour avancer de plusieurs colonnes vers la droite, c'est-à-dire la tabulation (Tab) et les deux codes qui restent omniprésents de nos jours :

- le code de passage à la ligne suivante (*Line Feed* ou *New Line*) ;
- le code de saut de page (*Form Feed* ou *Carriage Return*).

Ils sont insérés dans tous les fichiers des applications pour marquer des fins de paragraphes ou la fin du fichier.

L'ASCII étendu, mais il reste calme

Il suffit d'un seul octet pour tous les codes numériques des lettres et des symboles de la table ASCII. Au départ, on ne pouvait utiliser que les caractères anglais. Il n'y avait pas de place pour les lettres accentuées du français, les signes spéciaux de l'allemand, etc.

C'est au début des années 1980 que le nombre d'utilisateurs d'ordinateurs personnels a augmenté dans les pays non anglophones. Ils ont alors réclamé de pouvoir écrire des textes avec leurs caractères spécifiques. Des accords ont été passés entre ceux qui étaient chargés de la norme ASCII et les organismes de différents pays en leur attribuant l'espace correspondant aux valeurs de 128 à 255, c'est-à-dire la deuxième moitié de la plage de possibilités d'un octet.

Mais c'était une situation temporaire : chaque pays avait sa propre page de code. Si quelqu'un envoyait un fichier écrit avec une autre page de code que celle du destinataire, ce dernier voyait des signes bizarres ou différents de ceux qui s'y trouvaient à l'origine.

De nos jours, la situation est quasiment réglée, surtout depuis que les programmes se rendent compatibles avec la norme universelle Unicode et UTF-8. C'est notamment le cas des pages Web. Il devient enfin possible de mélanger du chinois, de l'espagnol et du hindi dans un même paragraphe.

Mais ces possibilités vont largement au-delà des besoins que nous avons avec notre petite carte Arduino. Nous allons nous contenter de la table ASCII de base, et au départ uniquement des lettres en majuscules de l'anglais. Pas d'accent, pas de minuscules. Voici donc cette fameuse table ASCII que des centaines de programmeurs ont un jour ou l'autre cherchée sous les tas de papiers de leur bureau (Figure 11.6).

Dans la colonne des valeurs binaires, les zéros de gauche ne sont pas montrés. Après tout, les prix dans les magasins ne sont pas écrits **003,99 €** non plus.

Tu as remarqué la colonne **Octal** ? C'est une autre base de numération que certains programmeurs adorent. Huit est un multiple de deux, ce qui est une bonne chose. Pour la valeur 101 de la lettre A, c'est facile : une soixante-quatraine, aucune huitaine et un, ce qui fait bien 65 décimal.

Table ASCII (codes de 0 à 127)

Decimal	Hexadecimal	Binary	Octal	Char
0	0	0	0	[NULL]
1	1	1	1	[START OF HEADING]
2	2	10	2	[START OF TEXT]
3	3	11	3	[END OF TEXT]
4	4	100	4	[END OF TRANSMISSION]
5	5	101	5	[ENQUIRY]
6	6	110	6	[ACKNOWLEDGE]
7	7	111	7	[BELL]
8	8	1000	10	[BACKSPACE]
9	9	1001	11	[HORIZONTAL TAB]
10	A	1010	12	[LINE FEED]
11	B	1011	13	[VERTICAL TAB]
12	C	1100	14	[FORM FEED]
13	D	1101	15	[CARRIAGE RETURN]
14	E	1110	16	[SHIFT OUT]
15	F	1111	17	[SHIFT IN]
16	10	10000	20	[DATA LINK ESCAPE]
17	11	10001	21	[DEVICE CONTROL 1]
18	12	10010	22	[DEVICE CONTROL 2]
19	13	10011	23	[DEVICE CONTROL 3]
20	14	10100	24	[DEVICE CONTROL 4]
21	15	10101	25	[NEGATIVE ACKNOWLEDGE]
22	16	10110	26	[SYNCHRONOUS IDLE]
23	17	10111	27	[ENG OF TRANS. BLOCK]
24	18	11000	30	[CANCEL]
25	19	11001	31	[END OF MEDIUM]
26	1A	11010	32	[SUBSTITUTE]
27	1B	11011	33	[ESCAPE]
28	1C	11100	34	[FILE SEPARATOR]
29	1D	11101	35	[GROUP SEPARATOR]
30	1E	11110	36	[RECORD SEPARATOR]
31	1F	11111	37	[UNIT SEPARATOR]
32	20	100000	40	[SPACE]
33	21	100001	41	!
34	22	100010	42	"
35	23	100011	43	#
36	24	100100	44	$
37	25	100101	45	%
38	26	100110	46	&
39	27	100111	47	'
40	28	101000	50	(
41	29	101001	51	)
42	2A	101010	52	*
43	2B	101011	53	+
44	2C	101100	54	,
45	2D	101101	55	-
46	2E	101110	56	.
47	2F	101111	57	/
48	30	110000	60	0
49	31	110001	61	1
50	32	110010	62	2
51	33	110011	63	3
52	34	110100	64	4
53	35	110101	65	5
54	36	110110	66	6
55	37	110111	67	7
56	38	111000	70	8
57	39	111001	71	9
58	3A	111010	72	:
59	3B	111011	73	;
60	3C	111100	74	<
61	3D	111101	75	=
62	3E	111110	76	>
63	3F	111111	77	?
64	40	1000000	100	@
65	41	1000001	101	A
66	42	1000010	102	B
67	43	1000011	103	C
68	44	1000100	104	D
69	45	1000101	105	E
70	46	1000110	106	F
71	47	1000111	107	G
72	48	1001000	110	H
73	49	1001001	111	I
74	4A	1001010	112	J
75	4B	1001011	113	K
76	4C	1001100	114	L
77	4D	1001101	115	M
78	4E	1001110	116	N
79	4F	1001111	117	O
80	50	1010000	120	P
81	51	1010001	121	Q
82	52	1010010	122	R
83	53	1010011	123	S
84	54	1010100	124	T
85	55	1010101	125	U
86	56	1010110	126	V
87	57	1010111	127	W
88	58	1011000	130	X
89	59	1011001	131	Y
90	5A	1011010	132	Z
91	5B	1011011	133	[
92	5C	1011100	134	\
93	5D	1011101	135	]
94	5E	1011110	136	^
95	5F	1011111	137	_
96	60	1100000	140	`
97	61	1100001	141	a
98	62	1100010	142	b
99	63	1100011	143	c
100	64	1100100	144	d
101	65	1100101	145	e
102	66	1100110	146	f
103	67	1100111	147	g
104	68	1101000	150	h
105	69	1101001	151	i
106	6A	1101010	152	j
107	6B	1101011	153	k
108	6C	1101100	154	l
109	6D	1101101	155	m
110	6E	1101110	156	n
111	6F	1101111	157	o
112	70	1110000	160	p
113	71	1110001	161	q
114	72	1110010	162	r
115	73	1110011	163	s
116	74	1110100	164	t
117	75	1110101	165	u
118	76	1110110	166	v
119	77	1110111	167	w
120	78	1111000	170	x
121	79	1111001	171	y
122	7A	1111010	172	z
123	7B	1111011	173	{
124	7C	1111100	174	|
125	7D	1111101	175	}
126	7E	1111110	176	~
127	7F	1111111	177	[DEL]

Figure 11.6 : La table ASCII avec ses valeurs dans plusieurs bases.

Données ASCII (char)

Deux types de données sont spécialement prévus pour manipuler des caractères : le type char et le tableau de caractères.

Le type char est un entier sur 8 bits. Tu peux l'utiliser comme toute autre valeur numérique, mais il sert d'abord à stocker des caractères, comme ceci :

```
char maLettre = 'A';
lcd.print( maLettre );
lcd.print( char(65) );
lcd.print( 65 );
```

Les deux premiers affichages ci-dessus ont le même effet. Le troisième affiche la valeur numérique 65 et non la lettre A.

Nous avons déjà utilisé des tableaux de caractères et nous allons en voir d'autres dans la suite du chapitre.

La technique d'affichage

Un écran LCD utilise une invention géniale qui consiste à orienter des petits cristaux mobiles, parce que liquides, soit dans un sens, soit dans l'autre. Dans un sens, les cristaux sont invisibles, et dans l'autre, ils forment une tache noire. L'énorme avantage de cette technique est qu'une fois que les cristaux sont orientés dans un sens ou dans l'autre, ils y restent sans consommer d'énergie. L'afficheur LCD est donc tout à fait adapté aux appareils électroniques qui fonctionnent sur piles.

Pour afficher les symboles, l'afficheur LCD propose une matrice élémentaire, comme une mosaïque constituée de cinq points dans le sens horizontal sur sept points dans le sens vertical. Entre deux symboles, un espace vertical est réservé pour espacer les lettres. Il y a aussi en bas une ligne de points réservée au soulignement ou au curseur de positionnement.

L'utilisation de l'afficheur devient très simple : tu lui transmets la valeur numérique du signe que tu veux faire afficher et sa position d'affichage. L'afficheur reçoit cette valeur et s'en sert pour aller chercher le dessin dans sa table ASCII stockée en mémoire. Par exemple, pour la valeur 65, il va chercher la case correspondant au A majuscule et allume les points en conséquence. Voici la table des dessins de caractères de l'écran LCD que nous utilisons (Figure 11.7).

La lettre A est à l'intersection de la colonne 0100 et de la ligne 0001. Le A majuscule en ASCII correspond donc bien à la valeur décimale 65. Le quartet supérieur (Upper) s'écrit 0100 (seul le bit des 64 est à 1) et le quartet inférieur (Lower) vaut 0001.

Dans certains composants, on utilise aussi un codage spécial nommé codage BCD.

Figure 11.7 : Dessins des mosaïques de caractères LCD.

Le codage hybride BCD

BCD signifie *Binary Coded Decimal*. C'est une astuce qui consiste à coder en binaire chaque chiffre décimal sur quatre bits. Comparons :

Si j'écris la valeur 65 du A en binaire normal, j'obtiens 0100 0001, c'est-à-dire 1 fois 64 et 1 fois 1.

Si j'écris la même valeur 65 en décimal codé binaire, j'obtiens tout autre chose. Le 5 décimal devient 0101 (1 fois 4 et 1 fois 1) et le 6 devient 0110 (1 fois 4 et 1 fois 2).

Au premier abord, ce n'est pas très futé. En codant chaque chiffre décimal à part, on perd de la capacité de codage : après 99 décimal, on doit utiliser un troisième quartet alors qu'en binaire pur, on peut aller jusqu'à 255 avec deux quartets.

L'encodage BCD a malgré tout sa raison d'être : il simplifie certaines opérations dans les circuits électroniques.

Nous en savons maintenant assez sur la théorie de l'afficheur LCD. Passons à la pratique, en commençant par choisir notre composant principal.

Montage d'un écran LCD

Il existe de nombreux modèles d'afficheurs LCD qui fonctionnent à partir du même circuit électronique Hitachi. Le modèle exact n'est pas important. L'essentiel est que ton afficheur comporte 2 lignes de 16 caractères. Ceci dit, s'il en offre plus, mes projets fonctionneront aussi.

Là où les choses se corsent, c'est que les afficheurs LCD sont prévus presque tous pour recevoir la valeur à afficher en parallèle. Puisqu'un code ASCII est sur huit bits, il faut donc huit broches. Et ce n'est pas tout ! Voici tous les branchements à prévoir pour un écran LCD :

- huit fils pour les huit broches de données ;
- deux fils pour l'alimentation 5 V et GND ;
- deux autres fils pour alimenter le rétroéclairage de l'écran ;
- trois fils pour des signaux de commande ;
- au moins un fil pour contrôler le contraste.

Si je compte bien, cela fait seize fils à brancher dont onze qui vont occuper des sorties numériques de la carte Arduino. Autrement dit, il ne te reste ensuite plus que trois sorties numériques pour piloter un autre appareil, ou simplement allumer des diodes LED.

Voici par exemple le circuit que j'avais prévu de te demander de construire. On le trouve à environ 6 ou 7 euros. Quand j'ai vu que le module enfichable coûtait moins de 11 euros, j'ai changé d'avis. Tu constates que le chantier était bien avancé :)

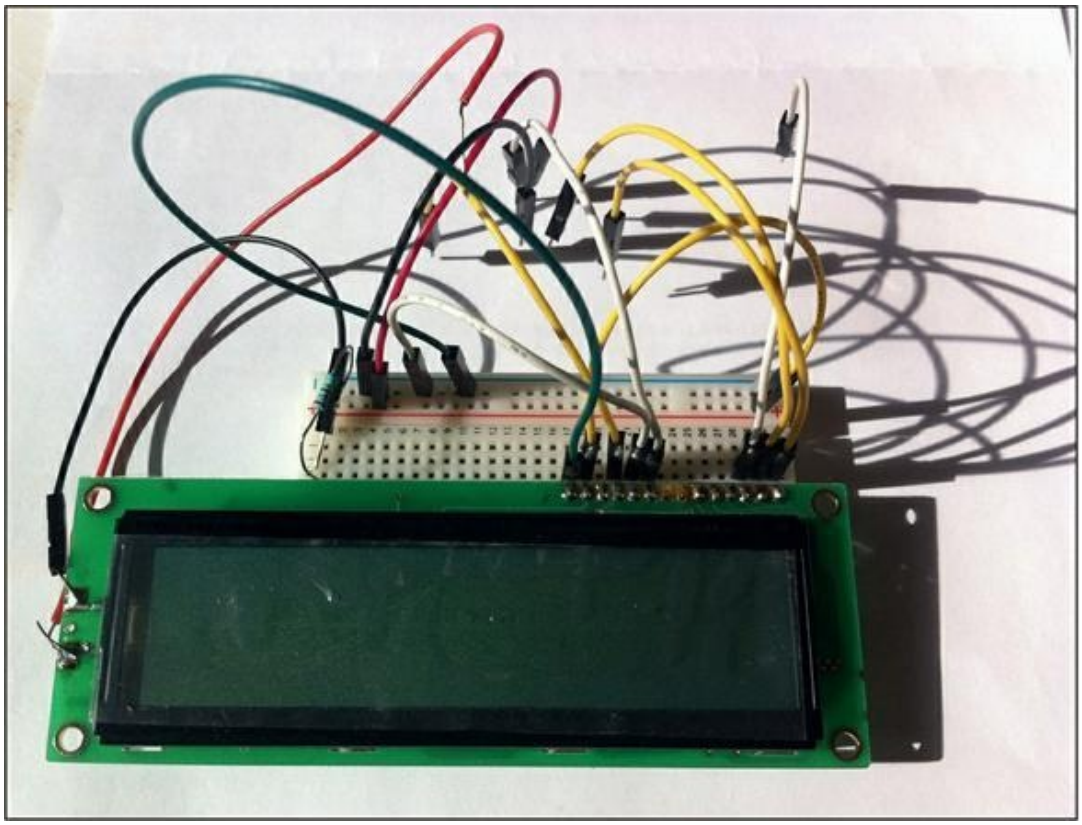


Figure 11.8 : Projet de câblage d'un écran LCD nu (abandonné).

Il y a deux solutions à ces soucis :

1. Tous les écrans LCD de cette gamme savent fonctionner en deux temps avec seulement quatre broches d'entrée pour les données. Il suffit de se mettre d'accord avec le programme pour qu'il envoie d'abord le premier demi-octet (un quartet), puis le second. C'est une sorte de liaison entre parallèle et série. On gagne quatre broches.
2. L'autre solution est radicale : elle consiste à souder l'écran LCD sur une carte qui possède les mêmes dimensions que la carte Arduino. Cette carte fille sera dotée sur le dessous du bon nombre de connecteurs

mâles pour pouvoir enficher la carte fille par-dessus la carte Arduino, en quelque sorte comme si elle protégeait sa mère. C'est pourquoi les cartes d'extension pour Arduino qui s'enfichent par-dessus portent en anglais le nom *shield*, qui signifie « bouclier ». La technique des deux demi-octets est bien sûr conservée.

La bonne nouvelle est que plusieurs fabricants proposent ce genre de module préconstruit. Il n'y a plus qu'à le payer, le déballer, l'enficher et téléverser le bon programme (avec une librairie appropriée, bien sûr).

Personnellement, cela me déçoit un peu de ne plus pouvoir voir mon joli mikon. Mais cela offre un tel avantage par rapport au câblage d'un spaghetti de fils que je me suis fait une raison.

Tu as donc deux possibilités au niveau du composant d'affichage :

- soit tu réunis un écran LCD nu, une pelote de fils et du courage ;
- soit tu t'équipes d'un module shield écran LCD.

Je te conseille fortement cette seconde solution, et je vais montrer comment monter le projet avec un modèle bien précis de carte fille : le module LCD de la société DFRobot sous la référence DFR0009 (Figure 11.9).

Figure 11.9 : La carte shield DFR0009.

Cerise sur l'écran, cette carte est dotée, en plus de l'écran, d'un potentiomètre pour régler la luminosité et de cinq boutons-poussoirs (plus un bouton Reset) pour de nouveaux projets que je te laisse imaginer !

Variantes ! Le circuit Toshiba HDC 44780 existe en version japonaise UA00 ou européenne UA02. La seule différence concerne la partie étendue des codes ASCII, c'est-à-dire les valeurs supérieures à 127. Il y a dans la version japonaise toute une série de kana.

Implantation de la carte fille

L'énorme avantage de la solution de la carte fille est qu'il n'y a aucun autre composant à ajouter, et aucun fil à connecter. N'est-ce pas magique ?

1. Déballe soigneusement la carte de son joli emballage. Tu vois que le dessous comporte de nombreuses broches. Elles vont s'enficher dans les connecteurs de la carte Arduino.

2. Prends la carte LCD par deux côtés dans une main et la carte Arduino dans l'autre. Observe l'espace entre les deux cartes. En tenant les deux cartes un peu de biais, positionne la première broche LCD dans le premier trou Arduino de chaque côté (donc la A5/SCL et la VCC qui est au-dessus de la DIGITAL 0). N'appuie qu'à peine.

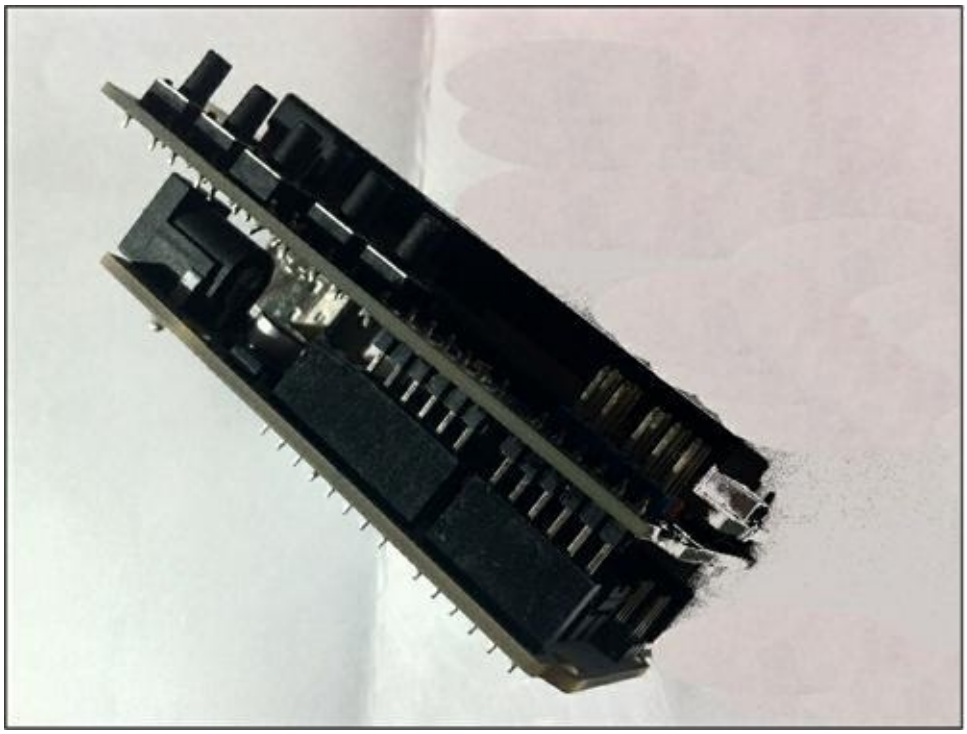


Figure 11.10 : Insertion de la carte shield.

3. Enfonce progressivement, sans forcer les deux rangées de connecteurs, jusqu'à ce que les deux cartes soient parallèles. Il est normal que les deux derniers contacts Arduino côté DIGITAL et côté ANALOG IN (côté prise USB) restent libres.

4. Et voilà ! Le montage est déjà terminé.

Voyons sans plus tarder ce qu'on peut tirer de cet afficheur.

L’écran LCD utilise les sept broches DIGITAL 4 à 10 ainsi que la broche analogique A0. Il ne faut pas essayer de les utiliser pour un autre composant.

Pour les très courageux

Si tu veux câbler un écran LCD nu, voici l’essentiel des branchements :

Signal	Contact de l’écran	Broche Arduino
Vss	1	Masse GND
VDD	2	Positif 5 V
Contraste (VE)	3	Curseur de potentiomètre
Register Select	4	Digital 8
Read/Write	5	Masse GND
Enable (E)	6	Digital 9
Donnée D4	11	Digital 4
Donnée D5	12	Digital 5
Donnée D6	13	Digital 6
Donnée D7	14	Digital 7
Anode Rétroéclairage	15	+5 V
Cathode Rétroéclairage à	16	Masse GND (Les broches 15 et 16 sont parfois

part.)

Pour le réglage du rétroéclairage, il faut un potentiomètre de 10 kohm. Une borne au 5 V, une borne à la masse et le curseur à la broche 3 de l'écran.

KristaLik le bavard

Comme premier projet d'affichage, je te propose de progresser en deux étapes :

1. Affichage de deux messages fixes occupant les deux lignes.
2. Affichage de deux messages avec une animation.

Analysons la réponse

Avant de coder, voyons s'il faut installer d'abord une librairie complémentaire. C'est un réflexe à adopter. Cet afficheur LCD est basé sur le circuit le plus répandu, le Toshiba HDC 44780. Tu seras donc heureux d'apprendre que la librairie qu'il faut a été acceptée par les créateurs Arduino comme librairie standard. Elle est donc installée dès le départ. Vérifions cela.

1. Dans l'atelier, ouvre le menu **Croquis**, puis le sous-menu **Inclure une bibliothèque** et, enfin, **Gérer les bibliothèques**. La fenêtre de gestion apparaît.
2. Dans la zone de recherche en haut à droite, clique puis saisis les premières lettres de `Liquid`. La liste se restreint (Figure 11.11).

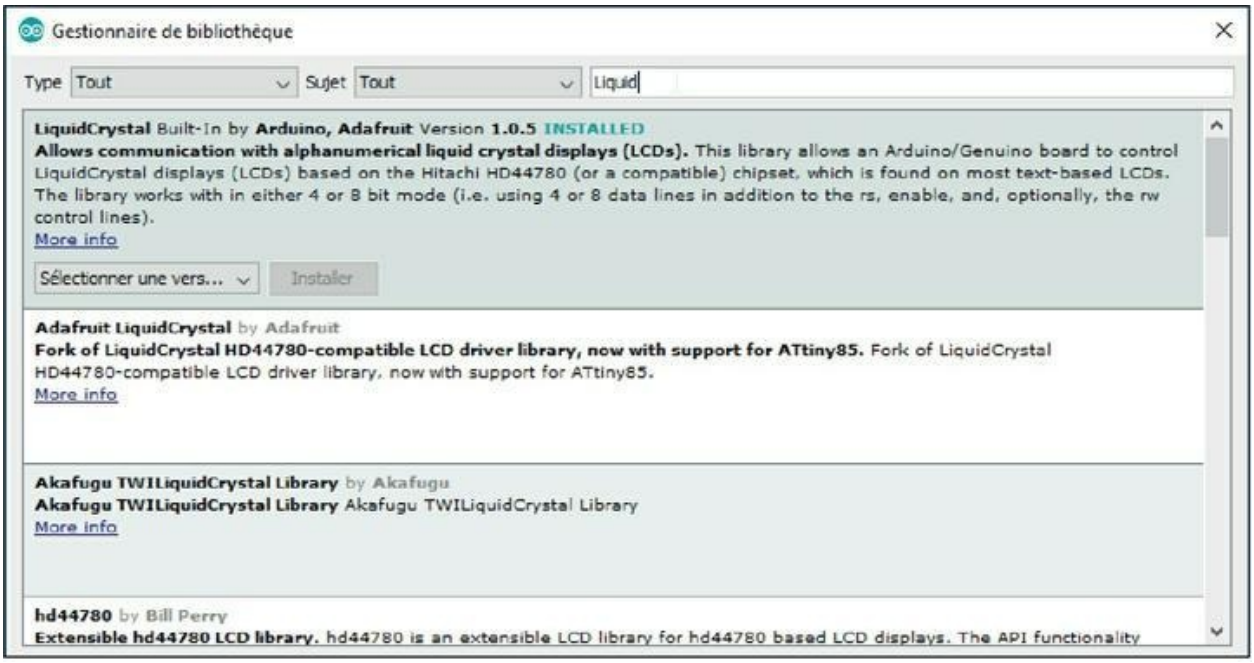

Gestionnaire de bibliothèque
Type Tout
Sujet Tout
Liquid
LiquidCrystal Built-In by Arduino, Adafruit Version 1.0.5 INSTALLED
Allows communication with alphanumerical liquid crystal displays (LCDs). This library allows an Arduino/Genuino board to control LiquidCrystal displays (LCDs) based on the Hitachi HD44780 (or a compatible) chipset, which is found on most text-based LCDs. The library works with in either 4 or 8 bit mode (i.e. using 4 or 8 data lines in addition to the rs, enable, and, optionally, the rw control lines).
More info
Sélectionner une vers...
Installer
Adafruit LiquidCrystal by Adafruit
Fork of LiquidCrystal HD44780-compatible LCD driver library, now with support for ATtiny85. Fork of LiquidCrystal HD44780-compatible LCD driver library, now with support for ATtiny85.
More info
Akafugu TWILiquidCrystal Library by Akafugu
Akafugu TWILiquidCrystal Library Akafugu TWILiquidCrystal Library
More info
hd44780 by Bill Perry
Extensible hd44780 LCD library. hd44780 is an extensible LCD library for hd44780 based LCD displays. The API functionality

Figure 11.11 : La librairie LiquidCrystal est bien installée.

Normalement, la librairie qu'il nous faut est mentionnée. Son nom exact est `LiquidCrystal`. Elle est indiquée comme de type `Built-In` (interne), ce qui confirme qu'elle est présente d'office.

Nous pouvons donc y faire référence dans nos textes sources par une directive #include.

Les fonctions essentielles

Pour exploiter un afficheur LCD à deux lignes, nous avons besoin de pouvoir faire trois opérations :

» Choisir où afficher le ou les prochains caractères en positionnant le curseur (invisible normalement). Nous utiliserons la fonction, pardon, la méthode d'objet nommée `setCursor()`.

» Demander l'affichage d'un texte, un ou plusieurs caractères à la fois, avec `print()`.

» Effacer tout l'écran, avec `clear()`.

Puisque la librairie est orientée objets, nous allons commencer par créer un objet à partir de la classe prédéfinie. Ici, elle se nomme `LiquidCrystal`, tout simplement. Pour l'objet, restons sobres : ce sera `lcd`.

Les trois méthodes seront bien sûr qualifiées par le nom de l'objet qui les possède, comme dans `lcd.setCursor()`.

Rédaction de KristaLik1

Comme tu sais le faire maintenant, saisis le texte source suivant ou bien charge-le depuis le dossier des exemples du livre. Il porte le nom suivant :

CH11A_KristaLiq1

Listing 11.1 : Texte source de KristaLik1

```
// KristaLiq1
// OEN170331

#include <LiquidCrystal.h>

LiquidCrystal lcd(8, 9, 4, 5, 6, 7);

void setup() {
  lcd.begin(16, 2);
}
```

```
void loop() {
  lcd.setCursor(0,0);
  lcd.print("Salut, Arduikidz !");
  delay(900);
  lcd.clear();

  lcd.setCursor(0,1);
  lcd.print("Ca va ti bien ?");
  delay(900);
  lcd.clear();
}

// BONUS
void test() {
  for (int i = 128; i < 256; i++) {
    lcd.print(char(i));
   delay(100);
  }
}
```

Visite guidée

La création de l'objet nommé `lcd` (son instanciation) est classique. Nous indiquons six numéros de broches en paramètres. Ici, pas de risque de se tromper puisque tu n'as rien câblé ! La lecture des paramètres te permet de confirmer quelles broches ne peuvent plus servir à autre chose.

```
LiquidCrystal lcd(8, 9, 4, 5, 6, 7);
```

Une chose à mémoriser est la syntaxe du positionnement de curseur. Les paramètres sont les suivants :

```
lcd.setCursor(colonne, ligne);
```

Le numéro de ligne ne peut être que soit 0, soit 1, puisqu'on compte à partir de 0. Je trouve illogique l'ordre des deux paramètres. On devrait d'abord choisir la ligne, puis la colonne. Qu'en penses-tu ?

J'ai laissé une routine en bonus. Pour la faire tourner, ajoute un appel vers `test()` dans la boucle principale. Dans cette fonction (ici, ce n'est pas une méthode), je demande d'afficher la conversion en type `char` de la valeur fournie. Vois ce qui se passe quand tu enlèves le type (qui devient ici un transtypeur).

Test du projet

Relis-toi et téléverse. Tu dois voir tes deux premiers messages (Figure 11.12).

Figure 11.12 : Kristalik1 en pleine action.

Un coup je te vois, un coup je ne te vois pas

Une petite astuce. Insère les trois instructions suivantes n'importe où dans la fonction principale pour masquer et réafficher ce qui est actuellement présenté à l'écran. Cette opération ne change rien au contenu.

```
lcd.noDisplay();
delay(900);
lcd.Display();
```

Dans une boucle de répétition, cela permet de faire clignoter l'affichage, par exemple pour attirer l'attention.

Du dessin ? Non, du texte animé

Voici une variante du projet qui affiche les messages un caractère à la fois, comme avec un effet de déroulé.

Le fichier source porte le nom suivant :

```
CH11A_KristaLiq2
```

Listing 11.2 : Texte source de KristaLik2

```
// KristaLiq2
// OEN170331

#include <LiquidCrystal.h>
#define LIG1 0
```

```
#define LIG2 1

LiquidCrystal lcd(8, 9, 4, 5, 6, 7);

char tcMsg[2][17] = {"Je suis le mikon",
                     " M'entends-tu ? " };  //
!

void setup() {
  lcd.begin(16, 2);
}

void loop() {
  lcd.setCursor(0,LIG1);
  lcd.print(tcMsg[0]);                  // !
  matrixer(1);
  delay(2000);
  lcd.setCursor(0,LIG2);
  lcd.print(tcMsg[1]);
  matrixer(0);
  delay(2000);
}

void matrixer(boolean sens) {
  for (byte yTour = 0; yTour < 16; yTour++)
{
    lcd.setCursor(yTour, (sens?LIG1:LIG2) );
// !
    lcd.print(" ");
    lcd.setCursor(yTour, (sens?LIG2:LIG1) );
    lcd.print(tcMsg[sens][yTour]);
    delay(100);
```

```
  }
}
```

Je déclare un tableau de deux chaînes de caractères avec une case de plus que la longueur maximale. Pourquoi ? Pour pouvoir stocker le caractère spécial qui marque la fin d'une chaîne en langage C. C'est la valeur 0. On appelle ces chaînes des chaînes à zéro terminal.

```
char tcMsg[2][17] = {"Je suis le mikon",
                     " M'entends-tu ? " };
```

Pour afficher chaque chaîne, j'indique seulement l'indice approprié :

```
lcd.print(tcMsg[0]);
```

Dans la fonction `matrixer()`, j'utilise à nouveau l'opérateur ternaire :

```
lcd.setCursor(yTour, (sens?LIG1:LIG2) );
// !
```

Le second paramètre sera égal à LIG1 si `sens` est différent de 0, et à LIG2 s'il est nul.

Pour accéder un à un aux caractères d'une chaîne, j'utilise le double indice :

```
lcd.print(tcMsg[sens][yTour]);
```

Lance un test. Tu vois que chaque caractère de la nouvelle ligne fait disparaître celui de l'autre ligne.

Figure 11.13 : KristaLik2 au milieu de ses affichages.

KristAlpha, pas si bête

Voyons maintenant comment produire de l'affichage de façon entièrement numérique, donc à partir des codes de caractères. Nous allons afficher tout l'alphabet en majuscules et en minuscules, et même quelques chiffres et signes de ponctuation.

Le fichier source porte le nom suivant :

```
CH11B_KristAlpha
```

Listing 11.3 : Texte source de KristAlpha

```
// KristAlpha
// OEN170331

#include <LiquidCrystal.h>
```

```
#define LIG1 0
#define LIG2 1
#define ADEB 65
#define AFIN 97

LiquidCrystal lcd(8, 9, 4, 5, 6, 7);
void setup() {
  lcd.begin(16, 2);
  Serial.begin(9600);
  analogWrite(10, 200);   // RÉTROÉCLAIRAGE
}

void loop() {
  alphabetiser(0);
  delay(1000);
  lcd.clear();

  alphabetiser(32);
  delay(1000);
  lcd.clear();

  alphabetiser(-32);
  delay(1000);
  lcd.clear();
}

void alphabetiser(byte ySaut) {
  byte ligne = 0;
  byte col = 0;
  for (byte yTour = ADEB; yTour < AFIN;
yTour++) {
    lcd.setCursor(col++, ligne);
```

```
    lcd.print(char(yTour + ySaut));
     if (yTour == ADEB+15) {
       ligne++;
       col = 0;
     }
     delay(200);
   }
}
```

L'appel suivant permet de contrôler la luminosité de l'écran :

```
analogWrite(10, 200);   // RÉTROECLAIRAGE
```

Dans la fonction `alphabetiser()`, une boucle affiche les caractères de toute une plage de codes ASCII. Je prends soin de passer en ligne 2 au bout de 16 positions.

Téléverse et teste. Tu dois voir les lettres en majuscules, les mêmes en minuscules, puis des signes et des chiffres.

Figure 11.14 : KristAlpha connaît les minuscules !

Essaye de voir ce qui se passe quand tu définis une plage trop grande, donc avec plus de caractères que la place disponible sur l'afficheur (32).

KristaProverbe, qu'on se le dise !

Je t'invite maintenant à une sorte de jeu du cadavre exquis. Le programme va combiner par tirage au sort le début d'une phrase à la fin d'une autre.

Bien sûr, nous allons déclarer une foule de chaînes de texte qui serviront de matière première.

Le fichier source porte le nom suivant :

```
CH11C_KristaProverbe
```

Listing 11.4 : Texte source de KristaProverbe

```
// KristaProverbe
// OEN170331

#include <LiquidCrystal.h>
#define LIG1 0
#define LIG2 1

LiquidCrystal lcd(8, 9, 4, 5, 6, 7);

#define MAXSUJ 11
char tcSujet[MAXSUJ][17] = {
  "Ne fais pas     ",
  "N'oublie pas    ",
  "Ne regarde pas  ",
  "Ne pense pas    ",
  "Ne raconte pas  ",
  "Raconte leur    ",
  "Ignore donc     ",
  "Ne nie pas      ",
  "Ne doute pas de ",
  "Ne dis pas      ",
```

```
  "Crois vraiment  " };

#define MAXOBJ 10
char tcObjet[MAXOBJ][17] = {
  "ce que je fais. ",
  "ce que je vois. ",
  "ce que j'aime.  ",
  "ce que je dis.  ",
  "ce que je crois.",
  "ce que je pense.",
  "ce que tu penses",
  "ce que tu fais. ",
  "ce qu'ils font. ",
  "n'importe quoi. " };

byte bHasaS, bHasaO = 0;
byte yPrecSuj, yPrecObj = 0;

void setup() {
  lcd.begin(16, 2);
  Serial.begin(9600);
  randomSeed(analogRead(1));
}

void loop() {
  bHasaS = tirerAuSort(yPrecSuj, MAXSUJ);
  yPrecSuj = bHasaS;
  bHasaO = tirerAuSort(yPrecObj, MAXOBJ);
  yPrecObj = bHasaO;

  derouler(bHasaS, bHasaO);
  delay(1000);
```

```
  lcd.clear();

  Serial.print(" Sujet  : ");
  Serial.print(bHasaS);
  Serial.print(" Objet  : ");
  Serial.println(bHasaO);
}

byte tirerAuSort(byte prec, byte plafond) {
  byte bNbr;
  bNbr = random(0, plafond);
  while (bNbr == prec) {
    Serial.print(bNbr);
    bNbr = random(0, plafond);
  }
  return(bNbr);
}

void derouler(byte yChoiS, byte yChoiO) {
  byte yTour = 0;
  lcd.setCursor(0, LIG1 );
  for (yTour = 0; yTour < 16; yTour++) {
    lcd.print(tcSujet[yChoiS][yTour]);
    delay(50);
  }
  delay(500);
  lcd.setCursor(0, LIG2 );
  for (yTour = 0; yTour < 16; yTour++) {
    lcd.print(tcObjet[yChoiO][yTour]);
    delay(100);
  }
  delay(200);
```

```
}
```

En lisant le code source lentement, tu ne dois pas avoir de surprises. Quelques remarques cependant :

```
randomSeed(analogRead(1));
```

Ici, nous utilisons la fonction standard qui remélange les chiffres du générateur de nombres `random()` afin que le tirage soit moins répétitif. Comme valeur, nous lisons ce qui traîne sur une broche analogique flottante.

Nous réutilisons une fonction de tirage au sort dédoublonnante que nous avons déjà rencontrée dans le Chapitre 7, dans l'exemple bonus `CH07B_ Devinum2`.

La série d'affichages de mise au point vers le Moniteur série peut être supprimée. Elle te montre qu'en cas de tirage du même nombre, il y a retirage.

Figure 11.15 : Un proverbe synthétique.

Lance un téléversement et regarde le mikon s'essayer à l'art de l'écriture. Il y a plus de cent combinaisons possibles. Certaines sont étranges, d'autres fendantes. Tu peux en ajouter à ta guise, mais n'oublie pas d'augmenter les indices des tableaux **MAXSUJ** et **MAXOBJ** en conséquence !

N'hésite pas à varier la durée d'affichage. Tu peux, par exemple, garder la même phrase affichée pendant une minute avec :

```
delay(60000);
```

Revenons à un projet à usage pratique grâce à un autre composant : le module d'horloge.

Est-ce que le temps vieillit ?

Le mikon sait compter le temps qui passe. Nous nous en sommes servis avec la fonction `millis()` qui renvoie le nombre de secondes écoulées depuis la mise sous tension de la carte.

Le problème, c'est que le mikon (c'est un comble) n'a pas de mémoire au sens où il perd l'heure quand son alimentation est interrompue.

Pour disposer de l'heure sans devoir la régler à chaque démarrage, et pour avoir au passage aussi la date année/mois/jour, il faut un circuit spécialisé. On désigne ce genre de circuit sous le nom *RTC*, Real-Time Clock ou horloge temps réel.

Un des circuits d'horloge les moins chers porte la référence DS1307. Il se dérègle un peu au cours du temps (de quelques minutes par mois), mais c'est largement supportable. Pour en tirer profit, il faut lui ajouter quelques résistances et, surtout, un cristal de quartz, comme le mikon.

Plutôt que de se lancer dans la création du montage, je te propose de chercher dans le commerce un module d'horloge préfabriqué basé sur le DS1307. Il en existe de nombreux modèles. Voici les questions à te poser pour que ton circuit soit compatible avec mes exemples :

1. Le circuit d'horloge doit être un DS1307.

Le circuit DS3231 est beaucoup plus précis, car compensé en température, mais il est plus cher.

2. La sortie doit être de type I2C (sur deux broches SCL et SDA).
3. Le module doit comporter sur l'autre face un porte-pile pour recevoir une pile en forme de jeton CR2032 (tension de 3,6 V).
4. Si possible, les broches de connexion doivent être déjà soudées ou bien un connecteur femelle doit être présent.

Pour ce livre, j'ai utilisé le modèle TinyRTC (Figure 11.16). Les composants sont sur une face et le porte-pile sur l'autre. Tu peux voir les contacts de connexion. Les broches n'étaient pas soudées lors de l'achat sur ce modèle.

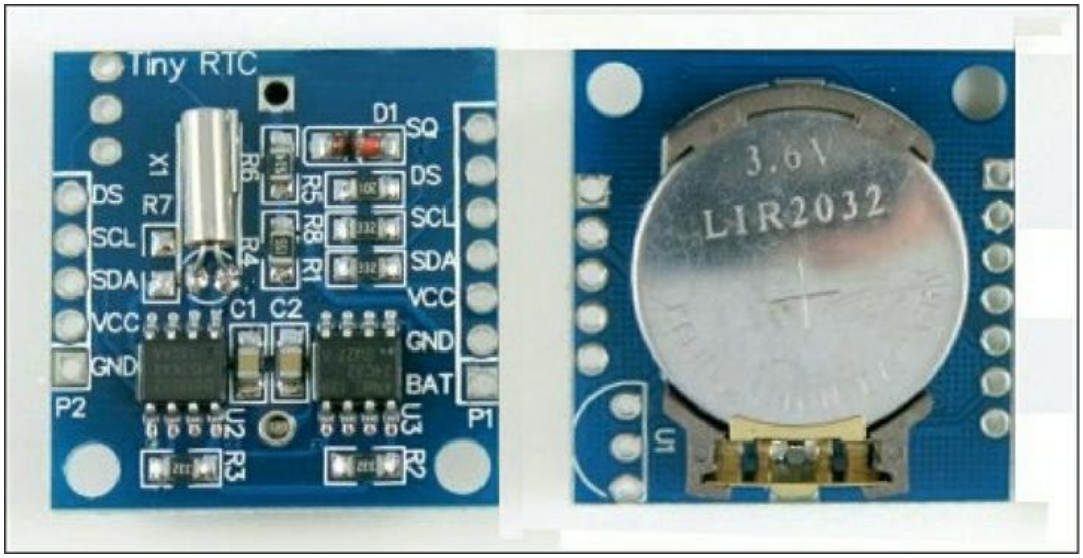


Figure 11.16 : Le module horloge TinyRTC basé sur le circuit du1307.

Montage

Il n'y a aucun composant supplémentaire à prévoir. Comme le circuit est minuscule, il faut néanmoins le fixer quelque part. Pour

le mien, j'ai enfiché le module dans un coin d'une petite plaque d'essai. Cela suppose que les broches de sortie sont soudées.

Si tu ne peux pas faire souder les quatre contacts, il te faut fixer le circuit sur un support, par exemple avec de l'adhésif, puis utiliser l'astuce (pas très glorieuse) indiquée dans le projet `Sonner` du Chapitre 9 pour faire les connexions.

Mais avant d'enficher ou de fixer la plaque, mieux vaut installer la pile bouton CR2032 car elle se trouve dessous. Le + doit être du côté restant visible.

Quatre broches seulement sont à connecter. Les autres peuvent rester inutilisées :

- les deux broches habituelles pour l'alimentation : GND (parfois marquée Vss) et 5 V (marquée parfois Vcc) ;
- les deux broches pour le dialogue I2C : SDA et SCL.

Un des avantages de la carte fille de l'écran est que plusieurs connexions de la carte Arduino sont remontées et donc disponibles par traversée sur la carte écran.

Observe le coin en bas à droite de la carte écran : tu y retrouves les broches des entrées analogiques. Le protocole I2C utilise justement les deux dernières, A4 et A5. Pour plus de confort, le fabricant a prévu pour chaque broche analogique une paire de broches d'alimentation. Tu peux ainsi rassembler les quatre arrivées de fils du module horloge dans ce coin de la carte (Figure 11.17).

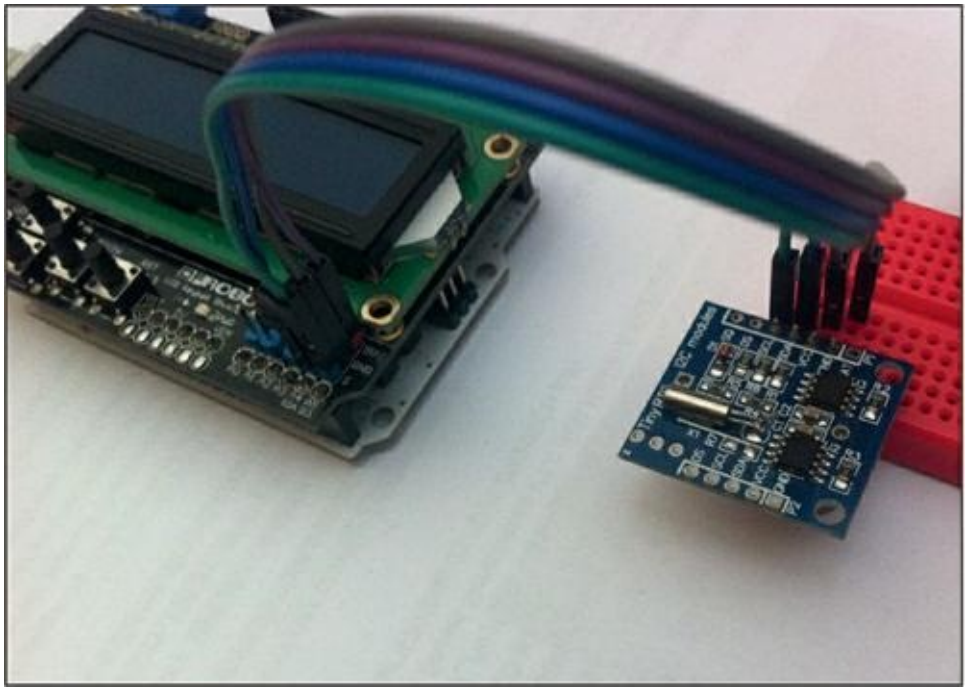


Figure 11.17 : Le module horloge fixé et connecté.

Voici les deux branchements pour les données :

- Le fil qui vient de la broche de données SDA du circuit horloge est à brancher dans la prise analogique A4.
- Le fil qui vient de la broche d'horloge SCL du circuit horloge est à brancher dans la prise analogique A5.

Je ne te fais pas l'affront de te rappeler qu'il faut comme d'habitude brancher les deux autres fils pour alimenter le module en énergie, soit le 5 V et la masse GND.

Et c'est tout. Passons à l'installation de la librairie qui va simplifier l'utilisation du module horloge.

Installation de la librairie

Plusieurs librairies sont disponibles pour gérer un circuit horloge DS1307. Toutes définissent au moins une classe qui permet de créer un objet de type temps. Cet objet sait fournir à la demande la date et

l'heure actuelles. Il suffit d'appeler la méthode appropriée de l'objet.

La librairie qu'il nous faut porte le nom `RTClib`.

1. Démarre l'atelier Arduino si nécessaire. Ouvre le menu **Croquis**, puis le sous-menu **Inclure une bibliothèque** et, enfin, **Gérer les bibliothèques**. La fenêtre de gestion apparaît.

2. Dans la zone de recherche en haut à droite, clique puis saisis les premières lettres de `RTC`. Plusieurs librairies sont proposées (Figure 11.18).

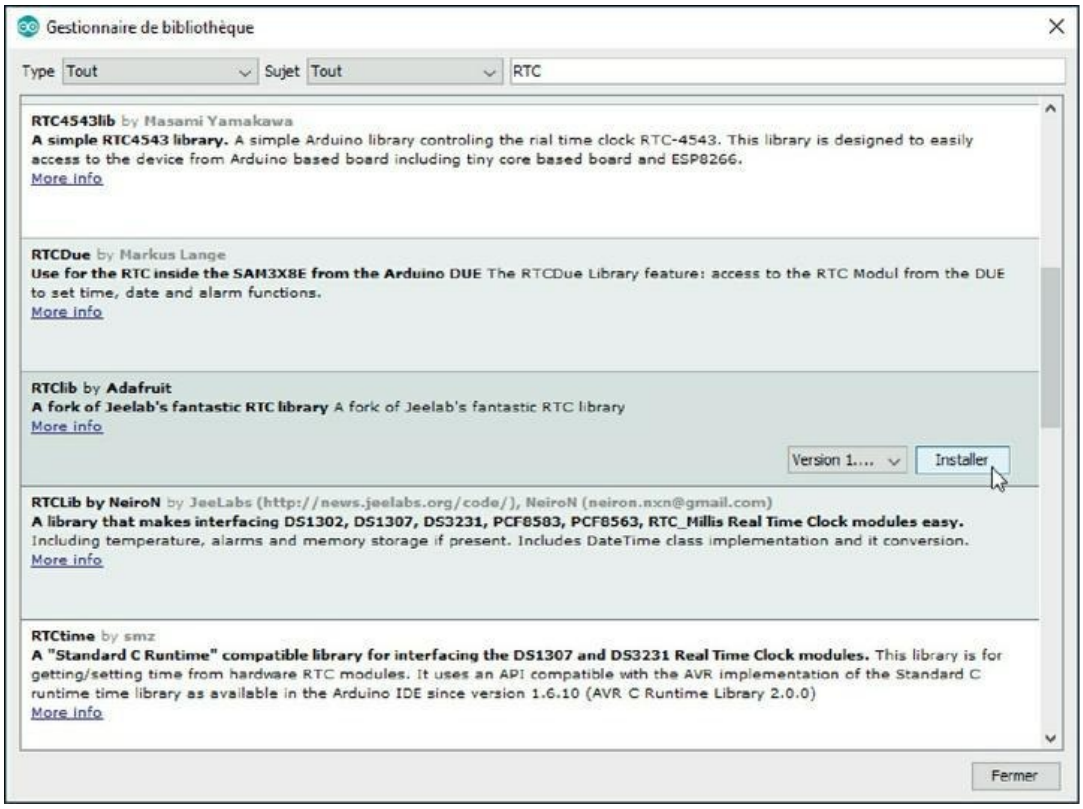

Gestionnaire de bibliothèque
Type Tout
Sujet Tout
RTC
RTC4543lib by Masami Yamakawa
A simple RTC4543 library. A simple Arduino library controling the rial time clock RTC-4543. This library is designed to easily access to the device from Arduino based board including tiny core based board and ESP8266.
More info
RTCDue by Markus Lange
Use for the RTC inside the SAM3X8E from the Arduino DUE The RTCDue Library feature: access to the RTC Modul from the DUE to set time, date and alarm functions.
More info
RTClib by Adafruit
A fork of Jeelab's fantastic RTC library A fork of Jeelab's fantastic RTC library
More info
Version 1....
Installer
RTCLib by NeiroN by JeeLabs (http://news.jeelabs.org/code/), NeiroN (neiron.nxn@gmail.com)
A library that makes interfacing DS1302, DS1307, DS3231, PCF8583, PCF8563, RTC_Millis Real Time Clock modules easy.
Including temperature, alarms and memory storage if present. Includes DateTime class implementation and it conversion.
More info
RTCtime by smz
A "Standard C Runtime" compatible library for interfacing the DS1307 and DS3231 Real Time Clock modules. This library is for getting/setting time from hardware RTC modules. It uses an API compatible with the AVR implementation of the Standard C runtime time library as available in the Arduino IDE since version 1.6.10 (AVR C Runtime Library 2.0.0)
More info
Fermer

Figure 11.18 : Installation de la librairie RTClib.

3. Sélectionne la librairie portant le nom `RTClib by Adafruit` et clique à droite son bouton **Installer**.

Avant de réaliser notre projet, il faut procéder à un premier réglage de l'horloge en faisant exécuter un programme dédié.

Réglage de l'horloge

Le circuit doit être synchronisé avec la date et l'heure réelles de ton lieu de vie. Je suppose que ton ordinateur est correctement réglé, car c'est de lui que nous allons faire récupérer les valeurs au moyen d'un petit programme que tu n'auras à exécuter qu'une fois après chaque changement de la pile (et encore). Et la pile dure plusieurs années.

Ce programme utilitaire est livré avec la librairie, parmi d'autres exemples.

1. Dans l'atelier, ouvre le menu **Fichier**, le sous-menu **Exemples**, puis la catégorie **RTClib** qui a dû apparaître tout en bas. Les quelques exemples sont listés.
2. Sélectionne celui portant le nom exact `ds1307` tout court.
3. Le texte source apparaît. En le parcourant, tu vois qu'il va afficher beaucoup de choses dans le Moniteur série. Le souci, c'est que la vitesse de dialogue est réglée à 57600 bauds au lieu des 9600 qui nous suffiraient amplement.

   Si tu ne coordonnes pas les vitesses, la mise à l'heure du circuit se fera tout de même, mais tu ne pourras rien lire à l'écran.

4. Si tu veux pouvoir lire les affichages, soit tu règles la vitesse dans la fenêtre du Moniteur série à 57600, soit tu modifies le programme.

   Je conseille de ne pas toucher à la vitesse dans le moniteur, d'autant que nous aurons une autre retouche à appliquer au code source.

5. Dans la première ligne de la fonction `setup()`, remplace ceci :

   ```
   Serial.begin(57600) ;
   ```

   par ceci :

   ```
   Serial.begin(9600) ;
   ```

6. Sauvegarde ta version retouchée du texte source `ds1307` sous un nom bien à toi. J'ai fourni la version modifiée, et traduite, tant qu'on y est, dans les exemples sous le nom `CH11C_ds1307`.

7. Lance un téléversement/exécution et ouvre la fenêtre du Moniteur série.

   Ne soit pas trop pressé pour ouvrir la fenêtre. Attends que le téléversement soit terminé ; sinon, tu vas créer un conflit d'accès à la liaison USB.

   Le temps d'ouvrir les yeux et le circuit ds1307 est déjà synchronisé ! En revanche, si tu lis bien les dates affichées et la promesse d'un commentaire du texte source, tu devrais repérer un petit souci.

Bon sang, mais c'est bien sûr ! Voyons le commentaire au début du second gros bloc d'instructions d'affichage :

```
// calculate a date which is 7 days and 30
seconds into the future
```

Si je traduis bien, il indique que nous allons calculer la date qu'il sera dans 7 jours et 30 secondes. Si tu calcules de tête, tu vas voir que ce n'est pas bon. Kézaco ?

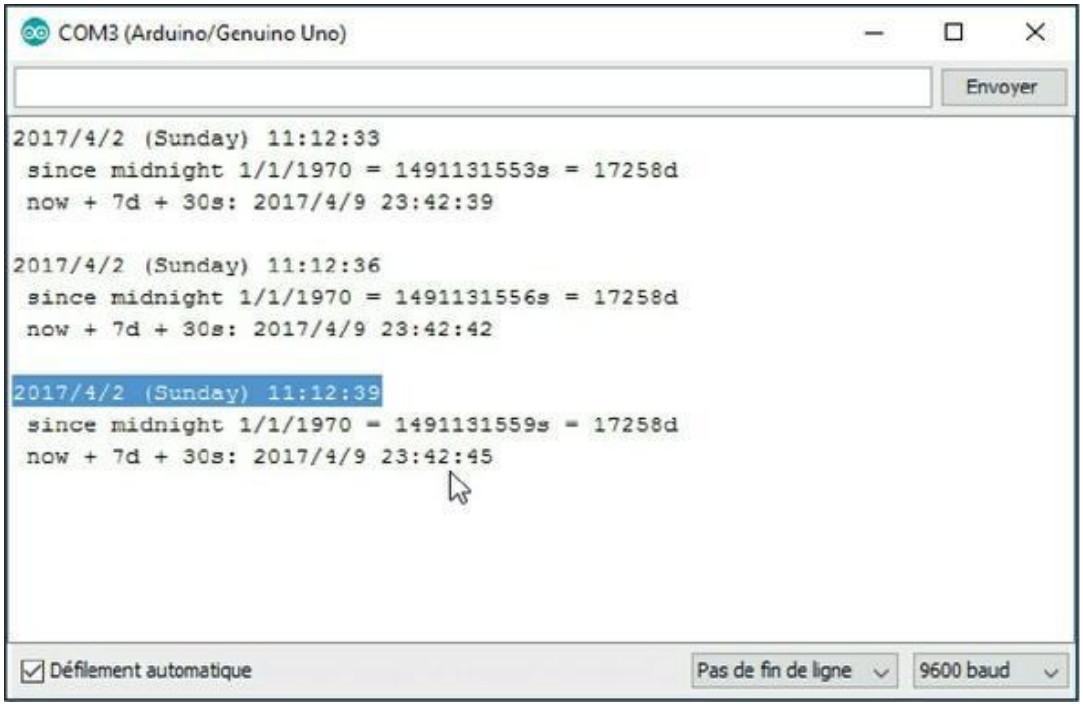


Figure 11.19 : Le calcul de la date future est bizarre...

Le calcul se fait dans l'instruction suivante. Elle crée un second objet de type date/heure en le réglant avec la date actuelle augmentée de ce que renvoie la méthode `TimeSpan()`. Et cette méthode a besoin de quatre paramètres d'entrée :

```
DateTime future (now + TimeSpan(7,12,30,6));
```

Ces valeurs 7, 12, 30, 6 te disent quelque chose. Le 7 est sans doute pour le nombre de jours. Mais le 12 ? Et si le 30 est pour les

secondes, à quoi correspond le 6 ?

Au travail, inspecteur Minute ! On passe en mode débogage. Tu sais que les librairies complémentaires sont stockées dans le sous-dossier `libraries` de ton dossier de travail Arduino ? Par exemple, sur ma machine, c'est dans `Mes documents`, en réalité ici :

```
C:\Users\ton_nom\Documents\Arduino\libraries\
```

Dans ce dossier, nous trouvons le fameux fichier d'en-tête (extension .h) qui est cité dans le texte source :

```
#include "RTClib.h"
```

Si j'ouvre ce fichier dans un éditeur (pas un traitement de texte, STP ! NotePad++ est gratuit et excellent), je peux faire chercher la méthode pour savoir comment elle doit être appelée. On parle aussi de signature de méthode. Voici ce que je trouve :

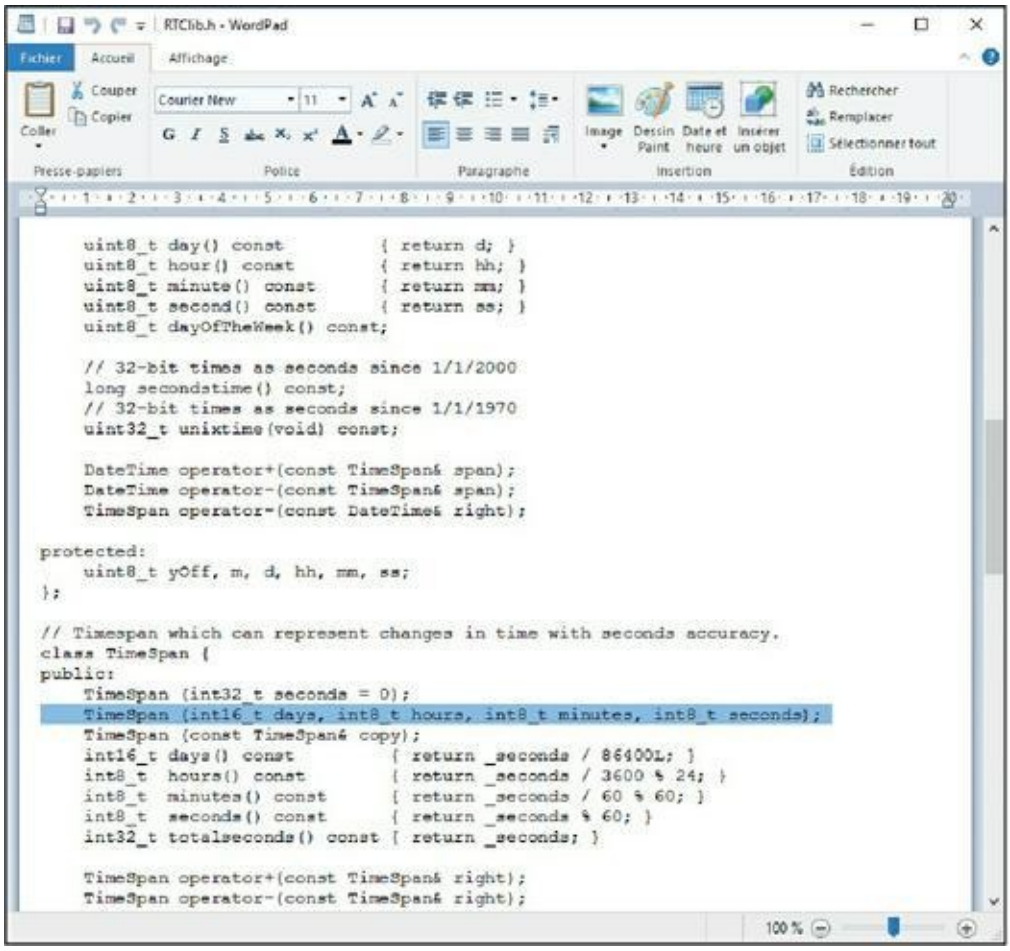


Figure 11.20 : Déclaration de la méthode TimeSpan().

Oui, il y a plusieurs déclarations pour la même méthode, mais expliquer cela nous mènerait vraiment à plus de 500 pages. Regarde la ligne sélectionnée :

```
TimeSpan (int16_t days, int8_t hours, int8_t
minutes, int8_t seconds);
```

D'accord. Il faut indiquer dans l'ordre le jour, l'heure, la minute et la seconde de décalage. L'exemple demande donc d'avancer dans le temps de 7 jours, 12 heures, 30 minutes et 6 secondes, pas de 7 jours et 30 secondes !

On se permet de corriger un exemple Arduino ? Allons-y.

1. Resauvegarde ton texte source par précaution.

2. Modifie alors l'appel à `TimeSpan()` comme ceci :

```
DateTime future (now + TimeSpan(7,0,0,30)
) ;
```

3. Compile, téléverse et ouvre le Moniteur série. Tout va mieux, non ?

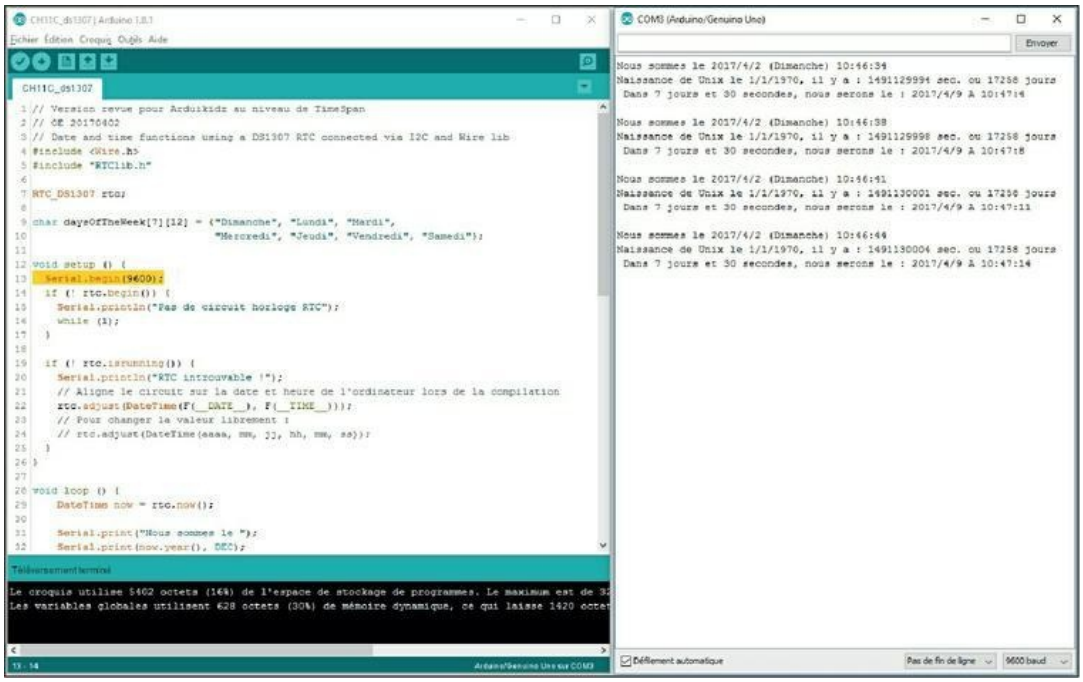


Figure 11.21 : Messages de réglage du circuit horloge.

Et maintenant, un projet. Nous allons combiner notre écran LCD et le module RTC (amusants, tous ces acronymes, quand on les connaît).

Le projet Tokante

Qui dit nouvelle classe, dit nouvel objet.

Ici, nous avons même deux nouveaux objets à instancier. En début de code source, nous créons l'objet principal qui va incarner le circuit horloge :

```
RTC_DS1307 rtc;
```

Dans la boucle principale, nous créons à répétition un objet de type date/ heure :

```
DateTime oDaH = rtc.now();
```

Nous répétons la création puisqu'il faut redemander l'heure sans cesse. C'est énervant, ce temps qui change sans cesse, non ? :)

La suite va consister à extraire un par un les « morceaux » de date et d'heure de cet objet.

Tu remarques la petite astuce pour parer à l'absence du zéro des dizaines pour les valeurs de 0 à 9 :

```
yMois = oDaH.month();
if (yMois <10)  lcd.print('0');
lcd.print(yMois, DEC);
```

Comme il n'y a qu'une instruction dans le bloc conditionnel, je me passe des accolades.

Le reste du programme est classique, si tu as lu ce livre dans l'ordre. On peut noter que les jours de semaine commencent le dimanche, à la manière anglo-saxonne (le dimanche correspond à l'indice 0 dans le tableau des noms de jours).

Listing 11.5 : Texte source de Tokante

```
// Tokante
// OEN170331

#include <LiquidCrystal.h>
#include <RTClib.h>

#define LIG1 0
```

```
#define LIG2 1

LiquidCrystal lcd(8, 9, 4, 5, 6, 7);
RTC_DS1307 rtc;

const byte RETROECL = 10;
char jourSemaine[7][5] = {"Dim.", "Lun.",
"Mar.", "Mer.",
                          "Jeu.", "Ven.",
"Sam."};
byte yMois, yJour, yH, yM, yS;

void setup() {
  lcd.begin(16, 2);
  Serial.begin(9600);
  if (! rtc.begin()) {
    lcd.println("Module RTC introuvable!");
    while (1);
  }
}

void loop() {
  DateTime oDaH = rtc.now();

  lcd.setCursor(0, LIG1);
  lcd.print(oDaH.year(), DEC);
  lcd.print('/');
  yMois = oDaH.month();
  if (yMois <10)   lcd.print('0');
  lcd.print(yMois, DEC);
  lcd.print('/');
  yJour = oDaH.day();
```

```
  if (yJour <10)   lcd.print('0');
  lcd.print(yJour, DEC);
  lcd.print(" ");

lcd.print(jourSemaine[oDaH.dayOfTheWeek()]);

  lcd.setCursor(4, LIG2);
  yH = oDaH.hour();
  if (yH <10)   lcd.print('0');
  lcd.print(oDaH.hour(), DEC);
  lcd.print(':');
  yM = oDaH.minute();
  if (yM <10)   lcd.print('0');
  lcd.print(oDaH.minute(), DEC);
  lcd.print(':');
  yS = oDaH.second();
  if (yS <10)   lcd.print('0');
  lcd.print(oDaH.second(), DEC);
  delay(1000);
//  lcd.clear();
}
```

Allez, on téléverse et on regarde le joli afficheur LCD. Tu sais créer une horloge (Figure 11.22).

Si tu vois s'afficher un avertissement (warning) pendant la compilation, pas de souci, d'autant qu'il ne concerne pas ton projet, mais la librairie. Un jeune Padawan ne va pas tenter de voir ce qu'il faudrait corriger dans le texte source d'une librairie.

Figure 11.22 : Ton horloge Arduino.

Pour clore le chapitre, je te propose une variante qui permet de gérer un autre fuseau horaire.

Listing 11.6 : Variation TokanteFuzo (extraits)

```
// TokanteFuzo Extraits
// Je rajoute trois lignes en début de texte
:
#define FUZO 6
// AJOUT
#define VILLE "BEIJING"
// MAX 7 car.!
char Tokyo[] = {196, 179, 183, 174, 179}; //
AJOUT
// ...
// ...
// ...
```

```
// La nouvelle fonction en fin de programme
:
void afficherHeure(DateTime DT, byte pays) {
  yH = DT.hour();
  yH += pays;                        // AJOUT
  // Tester si >23 => -24
  if (yH > 23) yH -=24;              // AJOUT
  if (yH <10)   lcd.print('0');
  lcd.print(yH, DEC);
  lcd.print(':');
  yM = DT.minute();
  if (yM <10)   lcd.print('0');
  lcd.print(yM, DEC);
  lcd.print(':');
  yS = DT.second();
  if (yS <10)   lcd.print('0');
  lcd.print(yS, DEC);
  if (FUZO) {                     // AJOUT
    lcd.print(" ");
    if (FUZO == 7) lcd.print(Tokyo); else
lcd.print(VILLE);
  }
}
```

Par rapport à la version de base, j'ai fait un peu de ménage. Cette opération se nomme la refactorisation. Au lieu d'une longue fonction principale, j'ai distribué le travail entre trois routines que la fonction appelle tour à tour. C'est plus lisible.

En début de texte, j'ai défini la constante FUZO que tu peux modifier pour changer de pays. Quand elle vaut 7, j'affiche le nom de Tokyo en kana. Quand elle vaut autre chose que 0, j'affiche le nom de la ville défini dans la ligne suivante.

On téléverse et on regarde. Tu remarques un test au début de `afficherHeure()`. En effet, s'il est 23 heures à Paris, en ajoutant 7 heures, on a bien l'heure au Japon, mais l'affichage indique 30 ! Voilà pourquoi il faut soustraire 24.

Figure 11.23 : Un fuseau horaire oriental et des caractères japonais !

Du fait que le module afficheur est doté de cinq boutons programmables, une extension du projet consisterait à détecter un ou plusieurs de ces boutons pour, par exemple, passer d'un fuseau à un autre. Saurais-tu t'en sortir sans explications ? Je donnerai peut-être la solution avec le fichier archive des exemples sur le site de l'éditeur.

Les boutons de l'afficheur DFR0009

Ce module utilise une astuce pour ne consommer qu'une seule broche analogique du mikon, la broche A0 (ou 14). Les boutons sont reliés à des résistances de valeur croissante, ce qui permet de les distinguer en mesurant la valeur convertie :

```
if (iValAna > 1000) return btnNONE;
 if (iValAna < 50)   return btnRIGHT;
 if (iValAna < 250)  return btnUP;
 if (iValAna < 450)  return btnDOWN;
 if (iValAna < 650)  return btnLEFT;
 if (iValAna < 850)  return btnSELECT;
```

```
    if (adc_key_in < 850)  return btnSELECT;
```

Grâce à return, dès qu'un test réussit, on quitte la fonction avec l'identifiant du bouton. Tu en sais assez pour t'éclater ! Installe un bloc switch où il faut et l'affaire est dans le sac.

Récapitulons

Dans ce chapitre, nous avons vu :

- les différents types d'afficheurs ;
- l'exploitation d'un afficheur LCD alphanumérique ;
- quelques jongleries avec des chaînes de caractères ;
- la recherche des causes d'un petit souci ;
- la création d'une horloge temps réel.

Chapitre 12
Un peu d'échologie

AU MENU DE CE CHAPITRE :

- **Le monde des sons qu'on n'entend pas**
- **Palper l'environnement**
- **Un instrument bizarre**

Après avoir fait un peu de littérature dans les précédents projets, retournons à ce vaste domaine qui est celui des ondes. Qu'elle soit électromagnétique ou acoustique, une onde permet de transporter de l'information. Et l'information, c'est la matière première de tout ordinateur !

Nous avons appris à contrôler :

- des ondes lumineuses ;
- des ondes infrarouges ;
- des ondes acoustiques.

Pour ces dernières, ce n'étaient que des ondes audibles par les humains. De même que l'on découvre à peine quelles formes de vie nous attendent au fond des gouffres océaniques, on commence à peine à aborder le vaste monde des ultrasons.

Moi, je dis ouïe !

Peut-on voir les yeux fermés ? Oui, avec ses oreilles ! C’est ce que font de nombreuses espèces animales, à commencer bien sûr par les chauves-souris. Ce petit mammifère émet de très courtes impulsions à des fréquences allant jusqu’à 100 kHz, soit cinq fois plus que la plus aiguë des notes que peut entendre un (jeune) humain. Tu devines que la chauve-souris est passée maître dans l’utilisation de ce sonar, car c’est ce qui lui permet de ne pas mourir de faim en attrapant des insectes en plein vol et par une nuit d’encre.

D’ailleurs, certains insectes auraient développé des capacités pour percevoir ces impulsions afin de changer de trajectoire au dernier moment. Pas bêtes, les petites bêtes !

L’acronyme sonar signifie Sound Navigation And Ranging, en français « navigation et mesure de distance par le son » . Le principe est quasiment le même que celui du radar, sauf que le radar utilise des ondes radioélectriques.

Les mammifères marins vont encore plus haut que notre Batman dans les aigus : le marsouin monte jusqu’à 150 kHz. Tous les cétacés se servent des ultrasons pour communiquer sur de très longues distances, car le son se propage très bien dans l’eau.

Plus près de nous, nos chiens entendent jusqu’à 40 kHz et nos chats, jusqu’à 60 kHz.

Les montages de ce livre qui émettent des sons utilisent un haut-parleur bon marché. Il ne risque pas de devenir capable d’émettre des sons aussi aigus.

Figure 12.1 : Quelques animaux ultrasoniques.

L'humain a bien compris l'intérêt des ultrasons. Depuis plusieurs dizaines d'années, les futures mamans passent des échographies. Les sous-marins utilisent un sonar pour repérer leurs ennemis. Les industriels font des contrôles non destructifs en envoyant des ultrasons dans du métal pour voir s'il cache des défauts. Il existerait même des canons militaires à ultrasons.

De façon moins belliqueuse, l'utilisation la plus répandue des ultrasons consiste à faire de la télémétrie, c'est-à-dire de la mesure de distance. Voyons comment créer notre propre télémètre.

Ton capteur d'ultrasons

Pour mesurer une distance, il faut émettre une impulsion dans une fréquence ultrasonore puis se tenir prêt à réceptionner l'écho qui rebondit sur l'obstacle. Il faut donc un émetteur et un récepteur.

Fonctionnement

Voici d'abord le principe de la télémétrie à ultrasons (Figure 12.2).

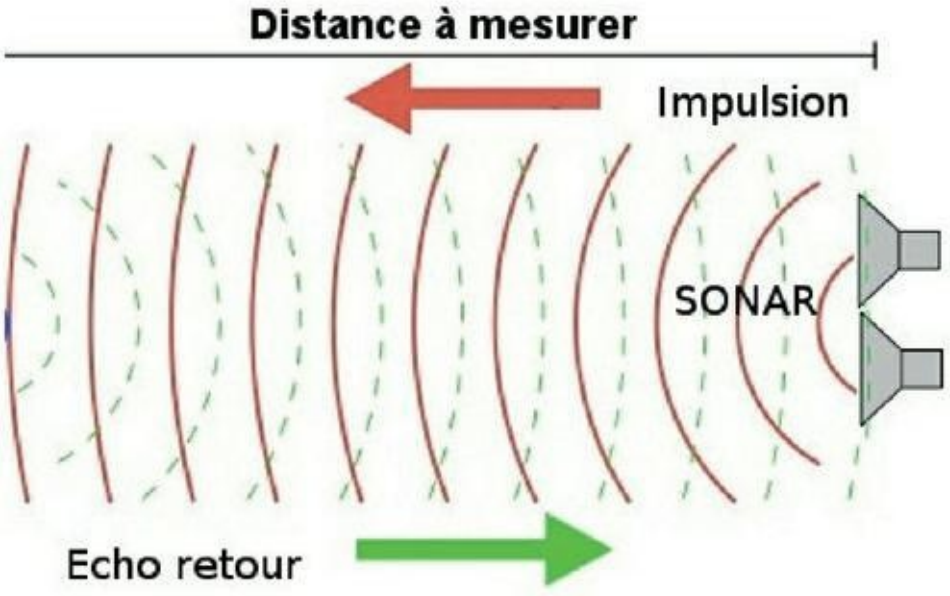


Figure 12.2 : Principe du sonar.

Chose incroyable, tu peux acquérir ton propre émetteur/récepteur à ultrasons pour quelques euros. Voici le modèle que je te propose d'apprendre à contrôler (Figure 12.3).

Figure 12.3 : Le capteur HC-SR05.

Comme pour d'autres capteurs des chapitres précédents, il y a un côté pour la partie capteur proprement dite (il y en a même deux ici) et un côté pour le circuit de contrôle.

Le modèle HC-SR05 offre cinq broches de sortie. Le SR04 en a une de moins, mais reste compatible. Si on met de côté les deux broches de l'alimentation (les fils rouge et noir), il en reste trois :

- la broche marquée TNG (qui devrait s'écrire TRG selon moi, pour *Trigger*, puisqu'elle sert à émettre l'impulsion) ;
- la broche marquée ECHO qui sert à recevoir l'écho après le rebond sur l'obstacle ;
- une troisième broche marquée OUT dont nous ne nous servons pas dans ce livre.

Je suis persuadé que la mention TNG est le fruit d'une faute de frappe chez le fabricant.

La séquence de commande du capteur est la suivante :

- Il doit recevoir une impulsion d'au moins dix microsecondes sur sa broche TRG, ce qui active le mode détection.
- Le module émet alors huit ondes carrées (haut/bas) à 40 kHz et attend le retour de l'écho.
- Lorsque l'écho revient, la broche ECHO bascule dans l'état haut et y reste autant de temps qu'il s'est écoulé entre l'envoi et le retour.
- Il ne reste plus, dans ton programme, qu'à mesurer ce nombre de microsecondes en surveillant le changement d'état de cette broche.

Le montage ne va donc pas être bien compliqué. C'est plutôt au niveau de la durée d'exécution des instructions qu'il faudra rester vigilant. Procédons donc au montage.

Montage de Telemaster

Deux possibilités te sont offertes selon que tu as ou pas déjà démonté l'afficheur LCD du précédent projet.

Montage sans l'afficheur

Si l'afficheur n'est pas installé sur la carte Arduino, il suffit de réaliser les branchements suivants :

1. Insère les cinq broches du capteur dans cinq rangées d'une plaque d'essai.

   Organise-toi pour que les deux cylindres du capteur soient dirigés vers l'extérieur de la plaque afin que rien

ne gêne le trajet de l'impulsion.

2. Branche un fil rouge entre le 5 V (VCC) du capteur et le 5 V de la carte Arduino.
3. Branche un fil noir entre la masse GND du capteur et l'une de celles de la carte.
4. Branche un fil blanc, jaune ou vert entre la broche d'émission TNG et la broche numérique `11` de la carte.
5. Branche un fil d'une autre couleur entre la broche ECHO du capteur et la broche `12` de la carte.

Le montage est terminé.

Montage avec la carte shield Afficheur LCD

Si la carte fille de l'afficheur est toujours en place, les branchements sont les suivants :

1. Branche les deux fils d'alimentation 5 V et GND où tu veux sur la carte LCD. Il y a plein de contacts disponibles.
2. Branche la broche TNG à la broche `D11` qui est rapportée en haut de la carte fille. Note que les numéros des broches du connecteur mâle ne sont pas dans l'alignement vertical des broches de la carte mère Arduino. Sers-toi du schéma illustré à la Figure 12.4.

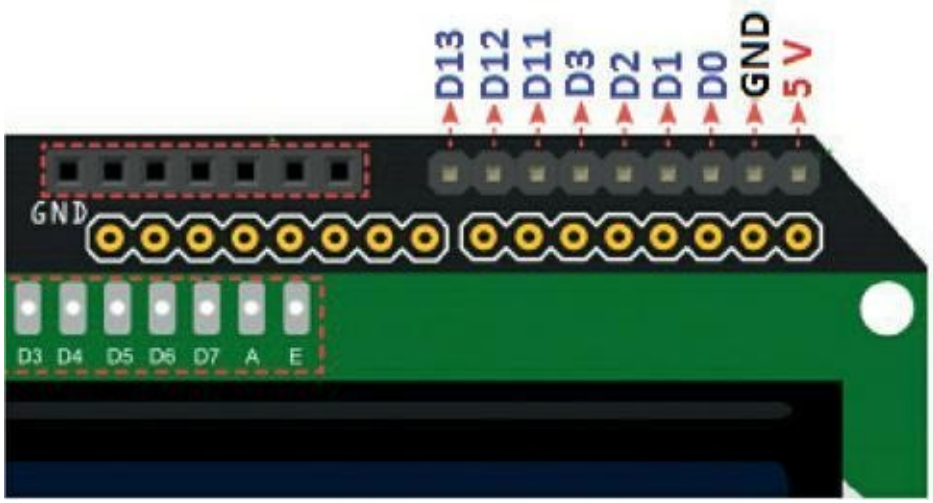


Figure 12.4 : Branchement du capteur à l'afficheur.

3. Branche enfin la broche ECHO sur la broche numérique `D12` de l'afficheur LCD.

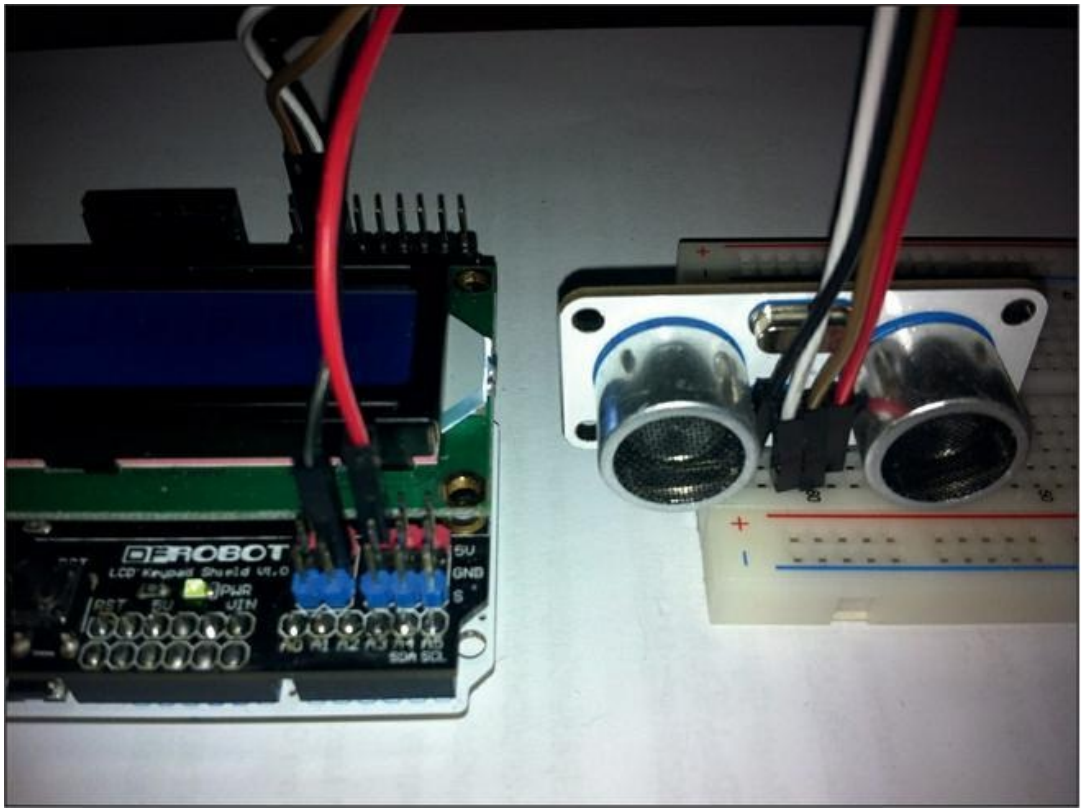


Figure 12.5 : Le montage de TelemasterLCD.

Passons maintenant au codage.

Rédaction du texte source de Telemaster

Pour exploiter l'émetteur/récepteur à ultrasons, il suffit d'écrire la mécanique qui correspond au principe que j'ai indiqué plus haut.

Trois étapes sont nécessaires :

1. Émission d'une impulsion très courte, ici de 10 µs.
2. Activation d'une fonction qui va se mettre à l'écoute du retour de l'écho en démarrant un chronomètre.
3. Calcul du délai de retour de l'écho. On obtient ainsi une valeur en microsecondes.

Dans l'air, l'onde sonore se déplace à environ 340 m/s. C'est pour cette raison que tu vois éclater les feux d'artifice avant de les entendre.

Un simple calcul permet de trouver que le son avance de 1 cm en 29 µs. Il ne reste plus qu'à diviser le nombre de microsecondes qu'il a fallu pour recevoir l'écho par cette valeur. Il ne faut pas oublier de multiplier la mesure par deux parce que nous avons chronométré le temps de l'aller et du retour. La distance est donc égale à la moitié de cette durée.

Il n'y a qu'une seule nouvelle fonction que nous allons découvrir dans le texte source qui suit. Il s'agit de `pulseIn()`.

```
fDistance = pulseIn(brRetourEcho, HIGH);
```

Son premier paramètre est le numéro de la broche de retour de l'écho ; le second paramètre est l'état qu'il faut surveiller.

Dans notre exemple, nous lui demandons de surveiller l'état haut (HIGH) de cette broche. Cela signifie que la fonction va se mettre en

attente jusqu'à détecter le passage de l'état bas à l'état haut. À ce moment, elle va commencer à faire tourner son chronomètre pour l'arrêter quand la broche redescendra à l'état bas.

J'ai choisi une variable à virgule flottante pour stocker la distance avec le type float.

```
float fDistance;
```

La préparation et l'émission de l'impulsion utilisent la fonction `digitalWrite()` :

```
digitalWrite(brEnvoiPulse, HIGH);
```

Saisis ou récupère ce texte source. Le nom proposé est `Telemaster`.

Listing 12.1 : Texte source de Telemaster

```
// Telemaster
// OEN 170402

#define brEnvoiPulse 11        // Broche TNG
du capteur
#define brRetourEcho 12        // Broche
ECHO du capteur
float fDistance;

void setup()
{
  Serial.begin(9600);
  pinMode(brEnvoiPulse, OUTPUT);
  pinMode(brRetourEcho, INPUT);
}
```

```
void loop()
{
  // Pour une impulsion de 10 micro-secondes
  digitalWrite(brEnvoiPulse, LOW);
  delayMicroseconds(2);
  digitalWrite(brEnvoiPulse, HIGH);
  delayMicroseconds(10);
  digitalWrite(brEnvoiPulse, LOW);

  // Palpation du monde
  fDistance = pulseIn(brRetourEcho, HIGH);
  Serial.print("Valeur brute en
microsecondes = ");
  Serial.println(fDistance, 0);

  // Vitesse du son de 340 m/s donc 29 us
par cm.
  fDistance = fDistance / (29*2);

  Serial.print(fDistance, 0); // Nombre de
décimales
  Serial.println(" cm");
  delay(300);
}
```

Test de mesure

Téléverse le projet vers la carte. Une fois le téléversement terminé, ouvre la fenêtre du Moniteur série.

Approche alors ta main du capteur. Lorsque tu fais varier la distance, tu dois voir la valeur affichée dans le moniteur changer

(Figure 12. 6).

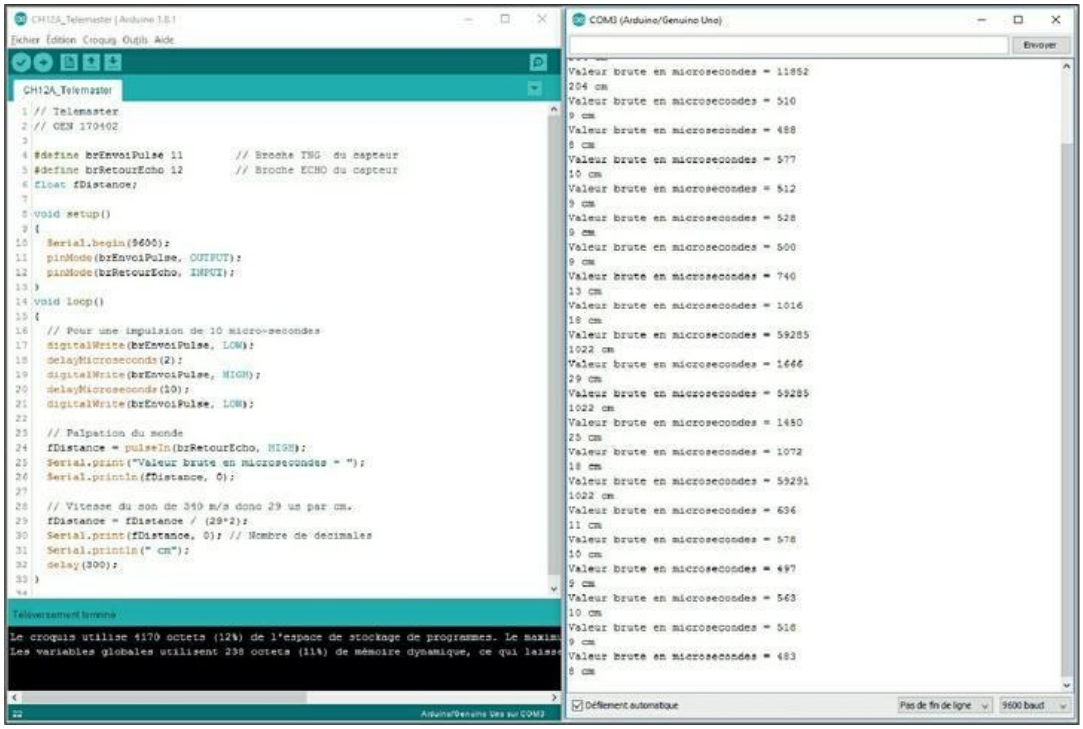


Figure 12.6 : Mesures de distance dans le Moniteur série.

La variante TelemasterLCD

Si tu as conservé en place la carte Afficheur LCD, tu peux très facilement faire afficher la distance sur cet écran. Il suffit d'ajouter les aménagements suivants :

- en début de listing, une directive d'inclusion de la librairie ;
- puis une instruction pour créer l'objet `lcd` ;
- dans `setup()`, une instruction pour démarrer le dialogue avec l'écran ;
- dans `loop()`, un appel à une nouvelle fonction que nous définissons ;

» enfin, le corps de cette nouvelle fonction qui va s'occuper de l'affichage sur l'écran.

Tout cela est rassemblé dans les extraits de listing suivants.

Listing 12.2 : Extraits du texte source de TelemasterLCD

```
// TelemasterLCD
// OEN 170402

// AJOUTER EN HAUT
#include <LiquidCrystal.h>           //
AJOUT
LiquidCrystal lcd(8, 9, 4, 5, 6, 7); //
AJOUT

// AJOUTER DANS setup()
  lcd.begin(16, 2);              // AJOUT

// AJOUTER A LA FIN DE loop()
  affLCD();                     // AJOUT

// AJOUTER EN BAS
void affLCD() {
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print("Distance en cm  :");
  lcd.setCursor(4,1);
  lcd.print( int(fDistance) );
}
```

Voici à quoi ressemble l'affichage d'une mesure sur cet écran (Figure 12.7).

Figure 12.7 : Mesure de distance dans l'afficheur LCD.

Rendre ton montage autonome

Puisque tu as un écran local et un capteur, tu peux rendre ton montage indépendant de l'ordinateur. Il suffit de te procurer un porte-pile (ou coupleur) pour une pile 9 V. La carte Arduino possède un connecteur rond que nous n'avons jamais utilisé. Il est du même côté que le connecteur USB. C'est le connecteur noir dans le coin inférieur gauche de la carte.

C'est là que tu peux brancher le jack standardisé de ton porte-pile. N'hésite pas à essayer ! Dans ce cas, pour économiser la pile, tu peux profiter du réglage du rétroéclairage en le réduisant un peu. Il suffit d'ajouter l'instruction suivante à la fin de la fonction de préparation `setup()` :

```
analogWrite(10, 50);                    //
RÉTROÉCLAIRAGE
```

Le second paramètre peut aller de 0 à 255.

Voyons maintenant comment transformer notre télémètre en un drôle d'instrument de musique.

Le projet Taprochpa

La transformation de notre projet Telemaster en instrument de musique est fort simple : nous y ajoutons un haut-parleur et contrôlons sa fréquence en fonction de la distance. Plus l'obstacle sera proche du capteur, plus le son sera aigu.

Montage du projet

Que tu aies ou non laissé en place l'afficheur LCD, il reste une broche numérique disponible à côté des 11 et 12 qui servent au capteur à ultrasons. C'est la broche 13. Voici comment procéder :

1. Raccorde un des fils du haut-parleur à la plaque d'essai dans la même demi-rangée qu'une résistance de 100 ohms environ (moins si c'est un haut-parleur de casque 32 ohms). De l'autre côté de la résistance, raccorde un fil à la broche numérique 13.
2. Raccorde l'autre fil du haut-parleur à la masse où tu en trouves une.

Le montage est terminé.

Codage

Au niveau du texte source, tu peux repartir de `Telemaster` ou de `TelemasterLCD` en l'enregistrant sous le nouveau nom `Taprochpa`.

Voici les aménagements à prévoir :

1. Ajout d'une directive pour définir la broche du haut-parleur, donc la D13.
2. Ajout dans la fonction de préparation d'une instruction pour régler la broche audio en sortie.

3. Ajout d'un bloc conditionnel à deux branches à la fin de la boucle principale. Cette partie conditionnelle est visible dans le Listing 12.3.

4. Définition d'une fonction du Chapitre 9 que j'ai réutilisée en la modifiant.

Je t'invite donc à apporter les modifications à ta copie du projet initial comme je te le propose dans le listing suivant (en italique).

Listing 12.3 : Texte source de Taprochpa

```
// Taprochpa
// OEN 170402

#define brEnvoiPulse 11        // Broche TNG
du capteur
#define brRetourEcho 12        // Broche
ECHO du capteur
#define yBroSON      13        // Broche
haut-parleur AJOUT
float distance;

void setup()
{
  Serial.begin(9600);
  pinMode(brEnvoiPulse, OUTPUT);
  pinMode(brRetourEcho, INPUT);
  pinMode(yBroSON, OUTPUT);            //
AJOUT SORTIE AUDIO
}
void loop()
```

```
{
  // Impulsion de 10 micro-secondes
  digitalWrite(brEnvoiPulse, LOW);
  delayMicroseconds(2);
  digitalWrite(brEnvoiPulse, HIGH);
  delayMicroseconds(10);
  digitalWrite(brEnvoiPulse, LOW);

  distance = pulseIn(brRetourEcho, HIGH);
  distance = distance / (29*2);
  Serial.print(distance, 0);
  Serial.println(“ cm  “);

  if (distance < 80) alarmer(distance);
// AJOUT
  else { tone(yBroSON, 440, 5); delay(500);
}  // AJOUT
}

// Recyclage de CH09B_Sonner3
void alarmer(int palpe) {
  int  iFreq = 40000 / palpe;
  tone(yBroSON, iFreq);
  delay(20);
  noTone(yBroSON);
}
```

Test audio

Téléverse le projet et ouvre tes oreilles. En fonction de la distance à laquelle tu places un obstacle, le son va varier en hauteur.

Tu peux tenter de faire de la musique comme avec une scie musicale. Le capteur est de temps en temps perdu : il indique la distance maximale lorsque les ultrasons sont mal renvoyés. Cela fonctionne mieux avec une feuille de papier épais ou un morceau de carton qu'avec ta main.

Récapitulons

Dans ce chapitre, nous avons vu :

- comment émettre une impulsion ;
- comment reccueillir la durée d'une impulsion ;
- comment asservir une fréquence à une distance.

Annexe
Mémento et futurs

Toute histoire a une fin, et tout livre aussi. Avant de nous quitter, je te propose quelques sujets pour t'aider à poursuivre ton chemin dans le fabuleux monde de la programmation.

Petit mémento du langage Arduino

Le langage de l'Arduino est inspiré du langage C, mais il n'en offre pas tous les éléments les plus récents, comme les fonctions lambda. Il suffit néanmoins au domaine d'emploi d'un mikon.

Fonctions internes

Ces fonctions sont toujours disponibles dès que l'atelier est installé.

Fonctions d'entrée/sortie (numérique, analogique)

Ce sont les fonctions fondamentales. Nous les avons toutes utilisées dans le livre.

```
pinMode(broche, mode)

digitalRead(broche)

analogRead(broche)
```

```
digitalWrite(broche, valeur)

analogWrite(broche, valeur) - PWM

tone(), noTone()

shiftOut(broche, BrocheHorloge, OrdreBit,
valeur)

pulseIn(broche, valeur)
```

Mesure du temps

Bien distinguer les deux couples : un travaille avec des millisecondes, l'autres avec des microsecondes.

```
millis(), micros()

delay(ms), delayMicroseconds(µs)
```

Arithmétique

La plupart de ces fonctions servent à faire des calculs. La fonction `map()` est intéressante : elle convertit une plage de valeurs vers une autre. Par exemple, si tu reçois sur une broche analogique une valeur entre 0 et 1 023, tu peux, avec cette fonction, compresser vers la plage entre 0 et 255, pour contrôler la luminosité d'une diode LED.

```
map(valeur, toLow, fromHigh, toLow, toHigh)

min(x, y), max(x, y), abs(x), constrain(x, a,
b),
```

```
pow(base, exposant), sq(x), sqrt(x),
sin(rad), cos(rad), tan(rad)

randomSeed(seed), random(min, max)
```

Communication

L'objet (en fait, la classe) `Serial` est le seul pour lequel il n'est pas nécessaire d'ajouter une directive #include en début de texte pour déclarer le nom de la librairie qui contient la classe.

Il sert, et tu le sais, à communiquer par liaison série ou USB, par exemple pour le Moniteur série de l'atelier.

Types de données

Les types de données int et long sont signés, sauf mention contraire unsigned. Autrement dit, le type int permet de stocker une valeur positive ou négative, ce qui divise par deux l'étendue des valeurs possibles.

```
boolean (1 bit)

char, byte (8 bits)

int (16 bits)

long
float

double
```

```
tableau[]

void
```

Je fournis dans le dossier **P4** des exemples un petit programme pour tester les tailles des types de données. Il porte ce nom :

```
zTypesData.ino
```

Voici ce qui s'affiche dans le Moniteur série quand ce programme est lancé :

```
sizeof(boolean)=   1

sizeof(byte)   =   1
sizeof(char)   =   1
sizeof(short)  =   2
sizeof(int)    =   2
sizeof(long)   =   4
sizeof(float)  =   4
sizeof(double) =       4

sizeof(int8_t) =   1
sizeof(int16_t) =   2
sizeof(int32_t) =   4
sizeof(int64_t) =   8

sizeof(uint8_t) =   1
sizeof(uint16_t) =   2
sizeof(uint32_t) =   4
sizeof(uint64_t) =   8

sizeof(char*) =   2
```

```
sizeof(int*) =   2
sizeof(long*) =   2
sizeof(float*) =   2

sizeof(void*) =   2

Plafond du type byte = 255
Plafond du type int = 32767
Plafond du type unsigned int = 65535
```

Constantes prédéfinies

Les deux premières lignes ci-dessous sont équivalentes aux valeurs numériques 1 et 0.

```
HIGH ¦ LOW

true ¦ false

INPUT ¦ OUTPUT

INPUT_PULLUP
```

Structures de contrôle

Nous avons utilisé certaines de ces structures, en tant que blocs conditionnels. Le mot clé continue sert à abandonner la suite des instructions en remontant au prochain tour d'une boucle de répétition for.

```
if,     if...else
```

```
for, while, do... while,

break, continue

switch case

return
```

Pour aller plus loin...

S'il fallait écrire un volume 2 pour ce livre, nous y découvririons de nouveaux capteurs et de nouveaux actionneurs.

D'autres capteurs

Parmi les capteurs qui t'attendent, je veux citer ceux-ci :

- **Détecteur de bruit** : pour savoir s'il y a du bruit ou pas. C'est en quelque sorte un microphone en mode tout-ou-rien.
- **Photorésistance** : sa valeur varie en fonction de la quantité de lumière reçue. C'est comme un capteur de couleur qui ne voit que du gris.
- **Capteur de pression** : pour détecter un appui ou un contact physique.
- **GPS** : pour recueillir des coordonnées spatiales.
- **Accéléromètre** : pour que le mikon sache s'il a la tête à l'envers et à quelle vitesse il se déplace dans

> l'univers. Ces modules se distinguent par le nombre de degrés de liberté.

D'autres actionneurs

Dès que le courant qu'il faut contrôler devient trop important pour les délicates broches du mikon, il faut utiliser un bras de levier.

Le transistor : un bras de levier

Le transistor est une sorte de sandwich de deux diodes collées tête-bêche.

Un petit changement sur une broche (la base) provoque un gros changement entre les deux autres (l'émetteur et le collecteur).

Le relais

Le relais prend le relais. Avec peu de courant pour changer son état (ouvert ou fermé), on peut contrôler un courant bien plus fort. En général, il ne faut pas brancher une sortie du mikon directement, mais passer par un transistor.

Le servo-moteur

Le servo-moteur est le composant essentiel de tout projet de robotique. Il se pilote simplement avec une impulsion dont la durée détermine l'angle de rotation du bras du servo. Et la librairie requise, `Servo`, est installée d'office.

Et de la communication !

Il existe enfin pour Arduino des modules Ethernet, WiFi, Bluetooth, etc. Il y a même une variante Arduino **Yun** qui réunit sur la même carte un mikon comme tu le connais et un autre processeur fonctionnant sous Linux !

Lis tes ratures !

De nombreux livres vont plus loin dans l'apprentissage de l'art de la programmation. Et les télécours (les MOOC, traduit en FLOT) fleurissent sur le web. Je me permets de citer un autre livre chez le même éditeur, du même auteur, qui s'enchaîne bien avec celui-ci : Programmer pour les Nuls (2e édition au minimum).

Les exemples Arduino préinstallés

Dans le sous-menu **Fichier/Exemples** de l'atelier Arduino, tu as accès à plusieurs dizaines d'exemples simples (catégorie Exemples inclus) que je te conseille de parcourir.

Vois notamment la catégorie **8. Strings** qui présente des fonctions de traitement de chaînes de texte à partir de la classe d'objets `String`.

Note : le texte source des exemples n'est pas encore francisé.

Je rappelle que lorsque tu installes une librairie, les exemples qu'elle contient viennent en général s'ajouter au sous-menu des exemples.

Ma sélection de revendeurs (où acheter)

Parmi les revendeurs sur la toile, j'ai établi une relation privilégiée avec la société **LetMeKnow** qui s'est organisée pour réunir et avoir en stock tous les composants requis sous la forme d'un kit au prix le plus juste.

Tu peux te rendre sur place à Paris ou commander via leur boutique web. Voici donc ma bonne adresse :

http://letmeknow.fr/shop/

Mais tu es libre de commander ailleurs. Voici ma liste restreinte de revendeurs sur la toile (en ordre alphabétique). J'ai commandé au moins une fois chez chacun et le service était satisfaisant.

- Conrad
- GoTronic
- LetMeKnow
- RobotShop
- Selectronic
- SemaGeek
- St Quentin Radio.
- VD RAM

Tous vendent par correspondance. Le fait de les citer ne m'engage évidemment pas.

Remerciements

Merci à mes chers et tendres Martine, Eva, Déreck et Léonard pour avoir supporté mon absence tous les soirs pendant tant de semaines.

Uu grand salut à toute l'équipe d'ingénierie qui me fait l'honneur de supporter mon humour : Jeroen, Melinda, Ralf, Stéphane, Sekou et tous les autres collègues.

Sommaire

Couverture

Programmer avec Arduino en s’amusant pour les nuls

Copyright

Introduction

- L’état d’esprit du livre
- Résumé de l’histoire (plan du livre)
- Pour réaliser les projets
- La communauté Arduino

I. Semaine 1 : Mise en route

- Chapitre 1. Quel est cet animal ?
  - Ton mikon est un ATmega328
  - C’est quoi son travail ?
  - Ta carte, c’est une Uno
  - Écouter et parler : les entrées et sorties
  - Un cerveau vide à la naissance ?
  - Séquence pratique : testons notre bébé
  - Langages de haut et de bas niveau
  - Récapitulons
- Chapitre 2. L’atelier du cyberartisan
  - Installation de l’atelier Arduino IDE
  - Installation de l’atelier

Premier démarrage de l’atelier IDE

Compilation de test…

… et téléversement de test

Autres outils

Récapitulons

Chapitre 3. Le tour de chauffe

Accéder au site du livre

Découvrons le moniteur de l’atelier

Récapitulons

II. Semaine 2 : Parler et écouter le monde

Chapitre 4. À l’écoute du monde en noir et blanc

Partir d’une feuille blanche (ou presque)

Projet 1 : détection sur une entrée numérique

Notre première variable

Récapitulons

Chapitre 5. À l’écoute du monde réel

Le projet Analo

Ta première fonction

Une instruction conditionnelle

Renvoyons nos valeurs

Un thermomètre analogique

Récapitulons

Chapitre 6. Parler, c’est agir

Une belle diode LED

Savoir faire tomber la tension

Un projet de luciole

Deux lucioles qui dansent (Luciole2)

Récapitulons

Chapitre 7. Donner une impulsion

La fonction analogRead()

Le projet Binar

Soyons logiques (Binar2)

Un plus un égale zéro ?

On peut compter sur lui (Binar3)

Un quartet en binaire

Voici Maître Devinum

Récapitulons

III. Semaine 3 : Son et lumière

Chapitre 8. Ta baguette magique

Le bon protocole

Un capteur d’ondes IR

Montage du projet BagMagik

Rédaction du texte source de BagMagik1

Compilation : erreurs ou avertissements ?

Comment s’utilise un objet ?

La version 2 de BagMagik

La version 3 de BagMagik

Récapitulons

Chapitre 9. Que personne ne bouge !

Théorie du capteur PIR

Pratique du capteur PIR

Une alarme audio

Retour au gardien

Testons nos oreilles

Récapitulons

Chapitre 10. Le voleur de couleurs

Des ondes lumineuses

La diode LED universelle

Ajoutons les couleurs secondaires

Tout est dans la nuance !

Le capteur de couleurs TCS 34725

Copier, c’est beau (VolKool2)

Récapitulons

IV. Semaine 4 : Le temps des lettres

Chapitre 11. Le temps liquide

Matrice de LED

Afficheur sept segments

Afficheurs LCD

Toujours avoir bon caractère

Montage d’un écran LCD

KristaLik le bavard

Du dessin ? Non, du texte animé

KristAlpha, pas si bête

KristaProverbe, qu’on se le dise !

Est-ce que le temps vieillit ?

Le projet Tokante

Récapitulons

Chapitre 12. Un peu d’échologie

Moi, je dis ouïe !

Ton capteur d’ultrasons

Le projet Taprochpa

Récapitulons

Annexe. Mémento et futurs

Petit mémento du langage Arduino

Pour aller plus loin...

Les exemples Arduino préinstallés

Ma sélection de revendeurs (où acheter)

Remerciements